Reversible Digital Watermarking In Digital Images with Recovery Scheme

by

Chung Farn Chon

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Technology (Honours)
(Information and Communication Technology)

MAY 2011

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Reversible Digital Watermarking

In Digital Images with Recovery Scheme

by

Chung Farn Chon

A project dissertation submitted to the

Information and Communication Technology Programme

Universiti Teknologi PETRONAS

in partial fulfilment of the requirements for the

BACHELOR OF TECHNOLOGY (Hons)

(INFORMATION AND COMMUNICATION TECHNOLOGY)

Approved by,

[illegible]

(JALE BIN AHMAD)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

May 2011

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

CHUNG FARN CHON

ABSTRACT

Digital watermark is a signal that contains information that is inserted into a digital medium such as images, videos, audios and etc. Digital watermarking is the process of embedding a digital watermark into the digital medium. Watermarking is commonly applied for copyright protection, copy protection and authentication of the digital media. A reversible watermark can provide all this with some extra features which include removal of the watermark safely from the watermarked media to return back the original media. Current practise of watermarking is that the watermark was directly embedded into the digital image by altering the image pixel bit. However, the image pixels may not be able to restore to its original value when the watermark is removed from the image. Memory Watermarking technique is proposed where the watermarking process are conducted in the memory. The image and watermark are read as memory bytes and the watermark are drawn to the image in the memory without affecting the physical image file. The watermarked image in the memory is displayed to the users while the physical file of the original image and watermark remain separated. The watermark can be added and removed with the restored image pixels one hundred per cent (100%) matched the original image pixels. A simple recovery procedure was built in to restored the original image if found the image of the watermark file was tempered.

ACKNOWLEDGEMENT

I would like to express profound gratitude to my supervisor, Mr. Jale bin Ahmad, for his invaluable support, encouragement, supervision and useful suggestion throughout this research work. His moral support and continuous guidance enabled me to complete my work successfully. His guidance in this research was invaluable where he has given me lots of guide and suggestion to improve my work to become better.

I also would like to express my gratitude to my fellow friends, Mr. Tan Kah Meng, Mr. Yap Eng Hoe, and many others for giving me ideas and support throughout this project. They had given me valuable feedback and suggestion to improve my research study especially in terms of the development of the system.

Next, I would like to thank my parents who support me endlessly throughout my studies in Universiti Teknologi PETRONAS. They had given me encouragement in this project and constantly asking me on the progress of this project so it would not affect my health and studies.

Finally, I would like to thank those who assisted me directly or indirectly in this project which is not mentioned above. Thank you for the support and assistance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background of Study

Digital watermark is a sequence of bits of signals that is inserted into a digital media such as images, videos, audios and etc. It may contain the information of the copyright ownership of the material. Digital watermarking on the other hand is the process of embedding a digital watermark into the digital media such as images. A digital watermark serves as a purpose to enforce and provide copyright protection on the digital media. With the digital watermark, owners can have their digital material protected and reduce the possibilities of copyright infringement.

1.2 Problem Statement

One type of digital watermark is a reversible watermark. A reversible watermark enables content owner to enforce their copyright of the digital media that they owned but at the same time to recover the original digital media that was watermarked (Feng, Chu, Lin, & Tsai, 2006). Unfortunately, data loss in terms of the image pixel bits is unavoidable although the watermark was removed from the watermarked media. Due to the embedding process which alters the bits of the images, the bits of the original image could not be recovered fully to get the same exact original image again after the watermark being removed. This loss of data can cause some wrong information especially in medical images, where it may lead to wrong diagnostic and wrong treatment.

1.3 Objective of Study

In this study, the objectives that need to be met are as follows:

1. To introduce memory watermarking as a new way of reversible digital watermarking.
2. To achieve one hundred percent (100%) matching with original image after watermark removal.

1.4 Scope of Study

The focus of this study is to create an application that uses memory watermarking technique to watermark an image and display it to the users. This study will only cover for the application of digital images such as Joint Photographic Expert Group (jpeg), Bitmap (bmp), Portable Network Graphics (png) and etc. It also should come with a simple recovery procedure to recover the image that was changed or corrupted.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

Digital watermarking is a process where hidden information is embedded into a digital media such as images, documents, videos and etc. The digital media is watermarked using an algorithm and resultant in a watermarked media with minimal distortion to the original media (Huang, Pan, & Hang, 2004). A watermark is considered as a noise embedded into the digital media which do not make significant changes to the original media as compared to the watermarked media.

Digital watermarking is commonly used in protecting the content ownership of the digital media. Watermark contains information of the content owner which is used to verify the ownership and authenticity of the watermarked digital media (Puhan & Ho, 2005). The watermark is embedded into the media through the watermark embedding process. Through the embedding process, the information of the ownership, copyright and etc. is embedded or inserted into the host media.

Watermark extraction is used to extract the watermark information out of the watermarked media. With the extraction of the watermark information, the ownership can be verified as this information cannot be altered. Digital watermarking is currently implemented for copyright of images as well as music and movies by the entertainment industry. Through digitally watermarked these media, the copyright holders are able to protect their copyright as well to be able to combat piracy and prosecute those who infringe their copyright.

2.2 Watermark Classification

2.2.1 Perceptible and Imperceptible Watermark

One of the classifications is perceptible watermark (Huang, Pan, & Hang, 2004). This classification of watermark is the watermark where it is visually visible in the digital media. Such digital media with perceptible watermark enable identification of the original content owner easier as the watermark is visually visible (Yang, Li, Sun, Yang, & Cing, 2008).

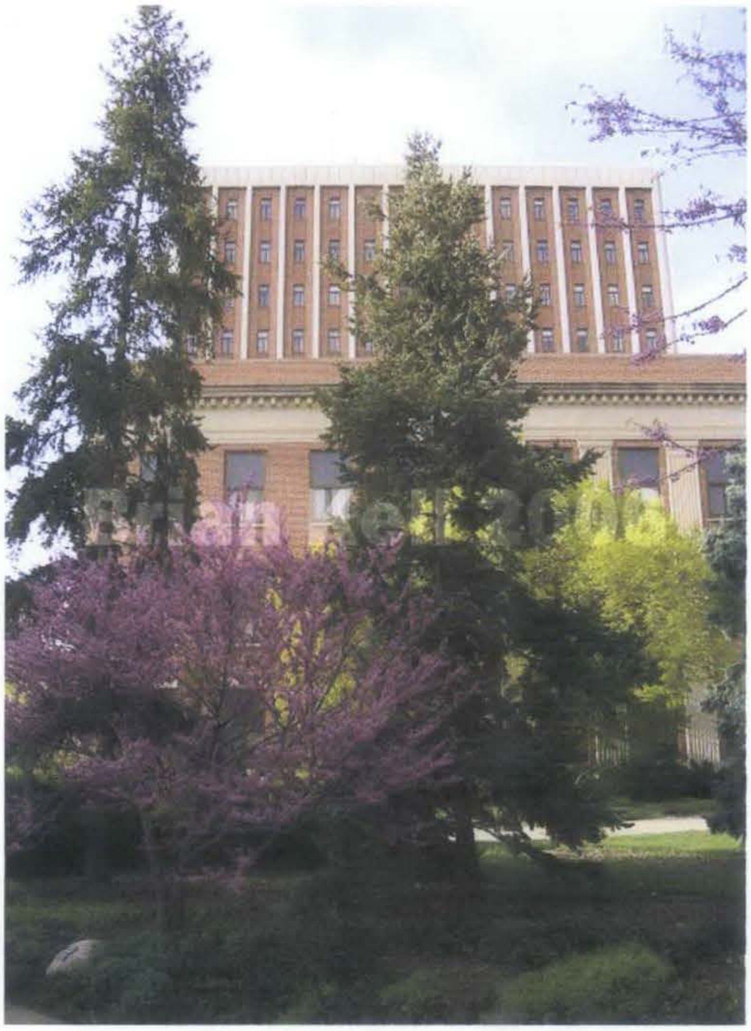


Figure 1: An image with visible digital watermarking – the text "Brian Kell 2006" is visible across the centre of the image (Source: http://en.wikipedia.org/wiki/Digital_watermarking)

As perceptible watermark is intended to be visible, the watermark must be difficult to be removed or resist falsification by unauthorized person

(Shih, 2007). The watermark need not be extracted to get the information as the visibility of the watermark can be used to verify the content owner of the watermarked media.

Figure 2: An imperceptible watermarked image

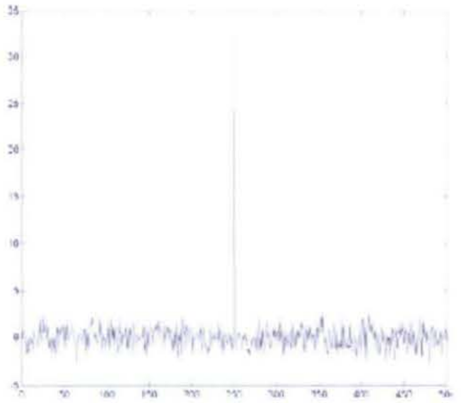


Figure 3: The result of the watermark detector using PSNR

Another classification of the watermark is imperceptible watermark (Huang, Pan, & Hang, 2004). Imperceptible watermark or commonly known as invisible watermark is not visually visible to the human eyes and it normally contain information on the copyright owner of the digital media (Bandyopadhyay, Paul, & Raychoudhury, 2010). The invisible watermark will contain the ownership, copyright information and etc. is embedded into the digital media which is invisible to the naked eyes but it will cause a slight distortion which is not visible or noticeable when comparing both the watermarked media and the original media.

Invisible watermark is embedded into the digital media by means of complex algorithm which will discuss in the embedding scheme of this report (Shih, 2007). An imperceptible watermark needs to be extracted using computer as to get the hidden information in the watermark from the watermarked media. With imperceptible watermark, no one are able to know the existence of the watermark without extracting it which it has then become one of the tools in order to thwart piracy especially in the video, audio or digital images.

2.2.2 Conventional and Reversible Watermark

There are two different techniques for watermarking which is conventional watermarking and reversible watermarking.

A conventional watermarking or also known as irreversible watermark is a watermark which could not be removed once it was embedded into the digital media (Feng, Chu, Lin, & Tsai, 2006). The watermark is embedded into the host media using a certain watermarking scheme or algorithm which will be discussed in the later part. The watermarked media can be then published online with the watermark that being embedded contains the copyright and ownership information. If the media was suspected to infringe the copyright, then the media can be checked using watermark detector such as using the peak signal-to-noise ratio (PSNR). If a watermark is detected, then information in the watermark can be extracted and the ownership of the media can be verified.

A reversible watermarking is a watermark that can be removed after it was embedded into the digital image to get back the original data (Zhou, Wang, Zhou, & Yu, 2010). When a reversible watermark is embedded into the host media, it should have the similar features as the conventional watermark. The difference between the reversible watermark and the conventional watermark is that the users were able to extract out or remove the watermark from the watermarked media (Feng, Chu, Lin, & Tsai, 2006). With this, the original media can be retrieved back along with the watermark and this is widely used in the medical and military industry (YongJie, Yao, Jeng-Shyang, & ShaoWei, 2005).

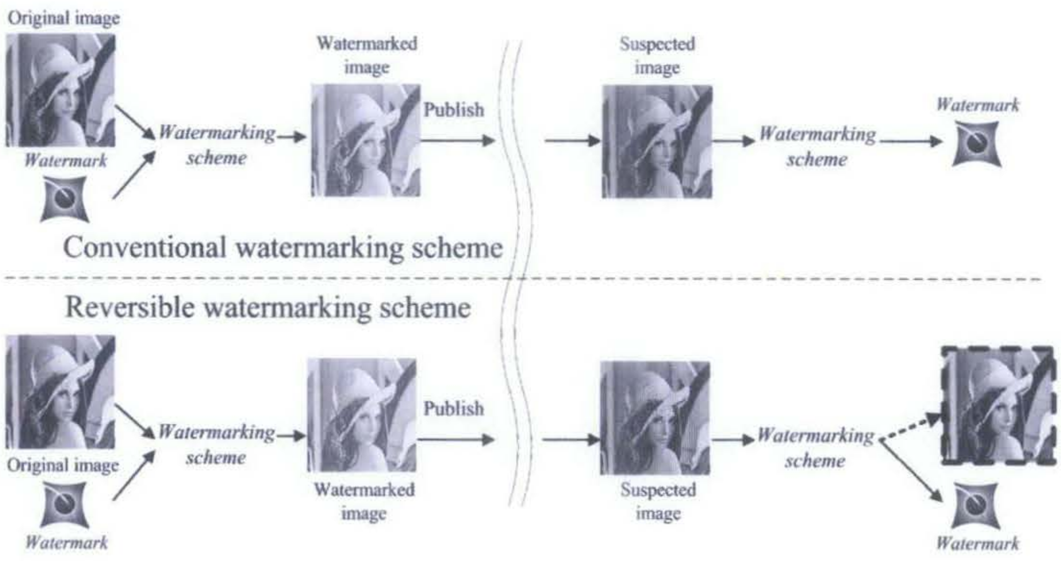


Figure 4: Conventional Watermarking vs. Reversible Watermarking (Source: (Feng, Chu, Lin, & Tsai, 2006))

2.3 Watermark Embedding Scheme

A digital watermark can be embedded into a host media by means of two different ways. These are the different type of algorithm that can be used to embedded watermark to the host media with each having its own way of embedding watermark along with its own properties.

2.3.1 Spatial Domain

Spatial Domain is the simplest form or algorithm for embedding watermark into a host media. A watermark is inserted into the host media for example an image where some of the grey value in the pixels of the image is changed (Shih, 2007). This is advantageous in terms of the small amount of computing power due to its low complexity needed as well as the easy implementation of the algorithm. According to Shih (2007), it is easily detected and is usually less robust against attack such as compression and noise.

2.3.2 Transformation Domain

The transform domain is introduced to address several weaknesses and limitations of the Spatial Domain. These watermarks are more robust against attack as compared to those embedded using Spatial Domain (Hsiang-Cheh, Jeng-Shyang, & Hsueh-Ming, 2004). It is commonly divided into three (3) which are Discrete Fourier Transformation (DFT), Discrete Cosine Transformation (DCT) and Discrete Wavelet Transformation (DWT).

2.3.2.1Discrete Fourier Transformation (DFT)

In DFT, an image is discomposed into set of orthogonal functions and can then transform the spatial intensity image into its frequency domain (Shih, 2007). This then is used to embed the watermark by selecting the adequate parts of the image. DFT commonly used to perform the phase modulation between the watermark and the host image (Jean-Luc & Stephan, 2000). Phase modulation is used in DFT instead of magnitude components for watermarking (Shih, 2007).

The general equation of DFT is defined as follows.

$$f = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} F(k_1, k_2) \cdot \exp[i \cdot 2\pi(x_1 k_1 / N_1 + x_2 k_2 / N_2)]$$

2.3.2.2Discrete Cosine Transformation (DCT)

DCT is a transform coding module for image and video coding standards such as MPEG and JPEG which makes this a popular watermarking algorithm. This is due to the compression standard set in JPEG and MPEG which this will make DCT-domain watermark are more robust (Jean-Luc & Stephan, 2000).

From the research done by Mei, Li, & Tan (2009), the DCT turn over the image edge to make the image transformed into the form of even function which is mathematically defined as

$$F(jk) = a(j)a(k) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f(mn) cos\left[\frac{(2m+1)j\pi}{2N}\right] cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

2.3.2.3Discrete Wavelet Transformation (DWT)

DWT is another type of algorithm that we can use to embed watermark into the host media such as image or videos. DWT become a key technique in source compression standard JPEG-2000. It offered a possibility of embedding in a compressed domain which others cannot achieve. Unlike DFT or DCT, DWT able to provide simultaneous representations for both the spatial and frequency interpretation.

In the research done by Terzija, Repges, Luck, & Geisselhardt (2004), the signal is decomposed or split into two part which are the high frequency and low frequency part. A series of high pass and low pass filter is used to analyse the high frequencies and the low frequencies respectively. The decomposition of the signal is mathematically expressed as follows:

$$y_{high}[k] = \sum_{n} x[k]g[2k-n]$$

and

$$y_{low}[k] = \sum_{n} x[k]h[2k-n]$$

The above procedure can be repeated for further decomposition and the output of both filters are known as DWT coefficients. The image is the decomposed into pyramidal structure with various band information: low-low frequency band LL, low-high frequency band LH, high-low frequency band HL and high-high frequency band HH.

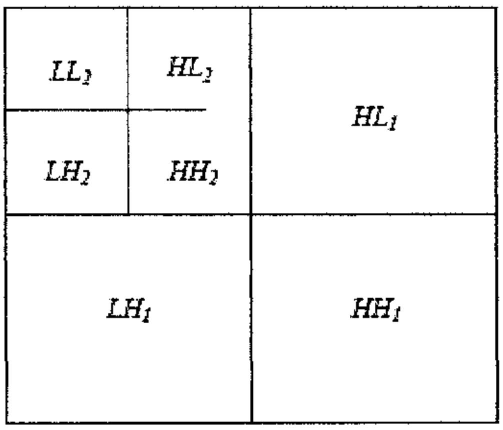


Figure 5: The pyramid structure.

2.4 Reversible Watermarking Embedding Scheme

A reversible digital watermark can be embedded using three (3) different methods in addition to the embedding scheme that commonly employed such as the frequency domain and spatial domain method.

The first is using data compression method. In data compression, the remainder is calculated based on the image quantified value. The remainder is then compressed using Context-Based, Adaptive, Lossless Image Coder (CALIC) lossless compression. The quantified image is then concatenated with the remainder value. Then, the watermarked image is obtained by adding the compressed data and watermark to the image. The retrieval of the watermark is by calculating the remainder back and the twelve (12) digits of the remainder are decompressed to sixteen (16) digit. The final four (4) bit is the hidden watermark.

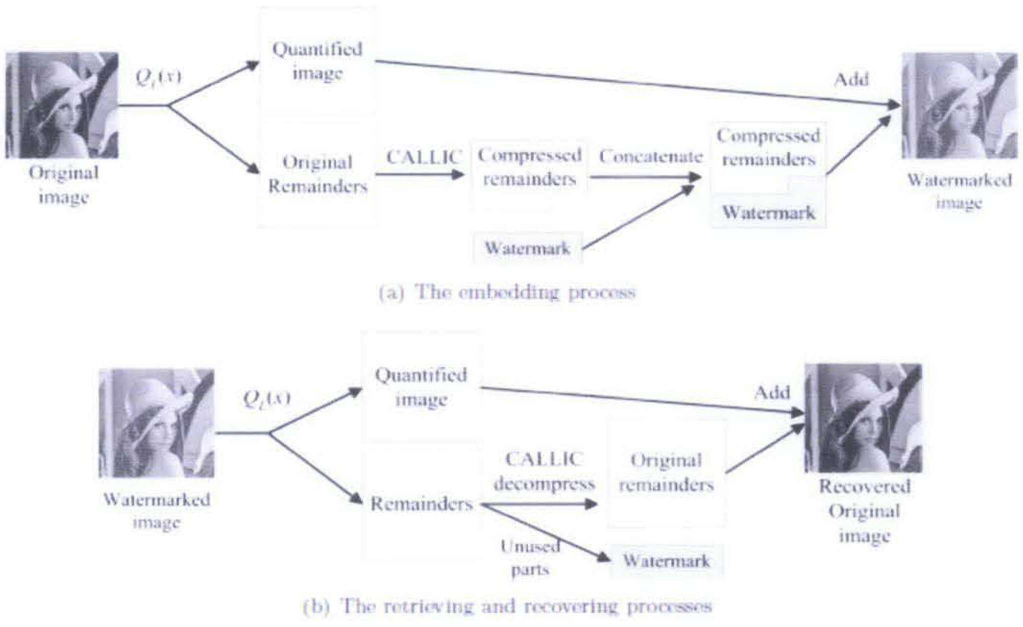


Figure 6: Data Compression Process

The next embedding scheme is the difference expansion. Difference expansion works by generating some small value to represent the features of the original image. It then expands the generated values to embed the bits of the watermark information. The watermark is embedded in the least-significant bit (LSB) and the watermarked image is reconstructed using the modified values. There are some problems exist in this scheme. Using Tian's Scheme (2003) which uses this method, there exists the possibility of data loss when the watermark is removed to restore the image to its original. Difference expansion is pixel based and the data loss will affect the block of the image pixels causing the value of the image pixel to be different but this data loss will not affect the next block. There is a location map which contains the additional information of the embedding of the pair of pixels. If this location map is destroyed, the mismatches of the pixel values are bound to occur.

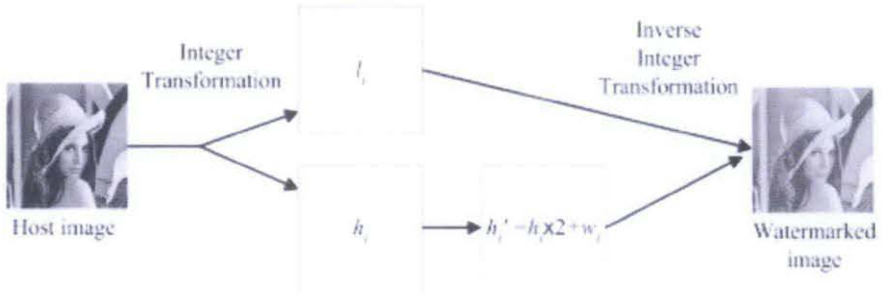


Figure 7: Tian's Scheme

Histogram bin exchanging is another method to embed the reversible digital watermark as proposed by Vleeschouwer et al. (2003). The embedding target is converted to histogram of a block. The original image on the other hand is segmented into several blocks of neighbour pixels. The shifting of the bins in histograms is done by shifting either the leftmost bins or rightmost bins of the histogram according to the corresponding bit of the watermark. The drawback of this method is that it can have slight distortion of the pixels. It can be due to the shifting of the bins which would cause the histogram to have extreme value or skewness.

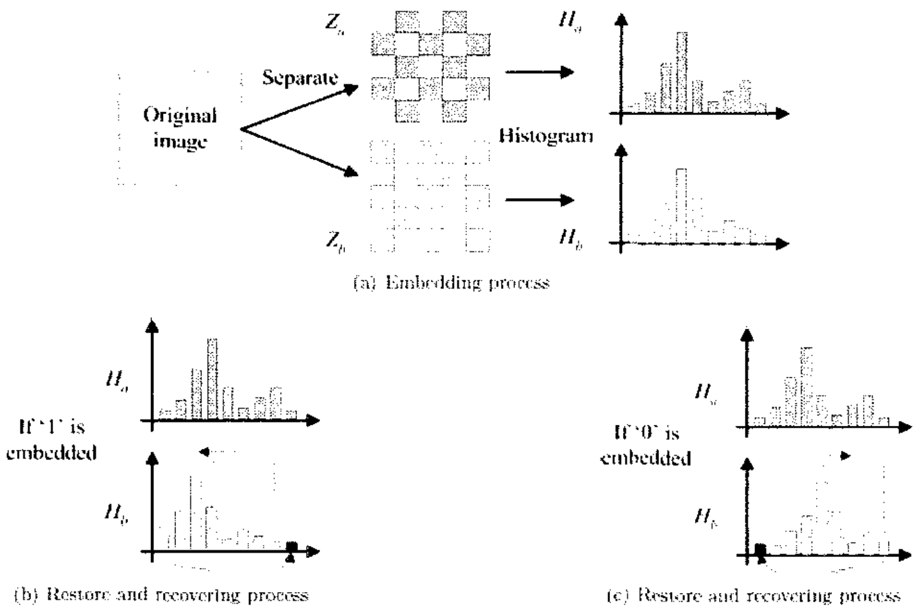


Figure 8: Histogram bin shifting process

2.5 Recovery Scheme

A watermarked image may suffer from attacks or even distortion which may cause the image to loss some information and even different from the original one. A recovery watermarking scheme would enable recovery of the modified media back to its original form.

An image can be divided into several blocks of specified sizes and the recovery information of the image is embedded into the least significant bits (LSBs) of each block of the images (Liew & Jasni, 2011) (He, Zhang, & Tai, 2009). From the research of Liew & Jasni (2011), the each block in the image is authenticated by comparing the hash value. If found any block was tampered, the recovery block will be used for the reconstruction or restoration of the tampered block image. Then, the watermark can be removed to restore it to the original state.

There are other type of recovery methods that other researcher proposed. DWT-SPHIT algorithm would also enable recovery of the digital image that was watermarked (Chen & Sun, 2010).

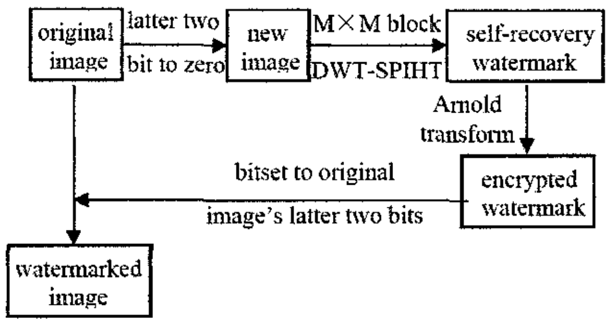


Figure 9: Embedding Process of Chen & Sun's Scheme

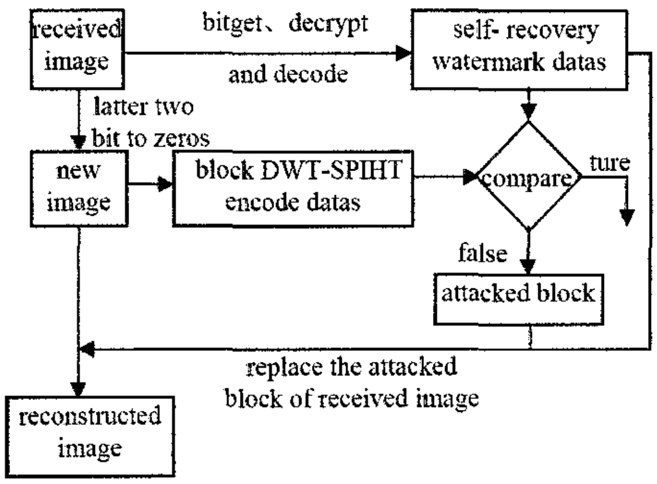

received image
bitget、decrypt and decode
self- recovery watermark datas
latter two bit to zeros
new image
block DWT-SPIHT encode datas
compare
ture
false
attacked block
replace the attacked block of received image
reconstructed image

Figure 10: Extraction and Self-Recovery Process of Chen & Sun's Scheme

Chen & Sun (2010) proposed that the original image is compressed with the block discrete wavelet transform and set partitioning in hierarchical trees (SPIHT) algorithm to get the recovery data due to the high performance and simplicity of the algorithm. The recovery watermark is then used to restore the attacked blocks by comparing the data in each block with the recovery watermark (Chen & Sun, 2010).

2.6 Secure Hash Algorithm

Secure Hash Algorithm also commonly known as SHA is one of the cryptographic hash functions that were published by the National Institute of Standards and Technology. SHA is an algorithm for generating one-way cryptographic secure hash. SHA-2 was currently the latest hash standard in SHA family where it succeeded from SHA-1. Collectively, SHA-2 consists of SHA-224, SHA-256, SHA-384 and SHA-512 hash functions which is named after the digest lengths (Secure Hash Standard, 2008).

SHA-512 uses six (6) logical function where each function will operates on 64-bit words and the result of each function is a new 64-bit word. SHA-512 uses a sequence of eighty constant 64-bit words where these words are the first 64-bits of the fractional parts of the cube roots of the first eighty prime numbers. The message to be hashed is padded where in SHA-512 case, the length of the message should be of a multiple of 1024 bits. The message undergoes some pre-processing where then message will be hashed using the functions defined (Secure Hash Standard, 2008).

$$\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)
\end{aligned}$$

$$\begin{aligned}
\textstyle\sum_0^{\{512\}}(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \\
\textstyle\sum_1^{\{512\}}(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \\
\sigma_0^{\{512\}}(x) &= ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x) \\
\sigma_1^{\{512\}}(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x)
\end{aligned}$$

Figure 11: SHA-512 Functions

An example of SHA-512 with the resultant hash value calculated is as follow. The resultant hash is 64-bits long.

String to be hash	: The quick brown fox jumps over the lazy dog
Result hash value	: 0x07e547d9586f6a73f73fbac0435ed76951218fb7d 0c8d788a309d785436bbb642e93a252a954f239125 47d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6

CHAPTER 3
METHODOLOGY

3.1 Requirement Gathering

The first stage of this research study is to gather the system requirement. A study on the previous work and research done by others researcher related to watermarking is done. This study on the related work is conducted to get the general idea of how the implementation of the reversible digital signature and the recovery scheme of the images. Through the research, algorithms will be analysed and reviewed to understand better how the watermarking works.

Through these studies on related works, the requirement of the system can be obtained and drafted out. With the gathered requirement, it will make sure that the system being build does not go out of scope as well able to achieve the objective. At the same time, how the system will work and the flow of the system is planned and drafted. This will give some initial kick start on how the system may look like and work in the final system.

Next, technical specification design of the project will be decided. It is essential to have a technical specification of the system as well as the design specification so that the project will not go off-track or having scope creep. This will help the developer to keep track of the functionality build in order to prevent this scope creep from occurring.

3.2 Equipment and Tools

The author makes use of these tools for the system development:

1. Microsoft Visual Studio Professional 2010
   Usage: Visual Studio is an integrated development tool by which the author used for the development of the system. The tool provides .NET Framework 3.5, C# and VB programming language for the development. It is used to compiles codes and creates the graphical user interface (GUI).

2. Matrix Laboratory (MATLAB) 2009a
   Usage: MATLAB is used for obtaining the pixel values of the images for the testing purposes. It enables the author to compare the image pixels values.

3. Some digital images
   Usage: The sample which is used for testing the system.

3.3 System Architecture and Design

3.3.1 System Architecture

In this study, the system consists of three (3) different modules which each provide different functionalities. These modules works together to enable the author to achieve the targeted objectives for this research study.

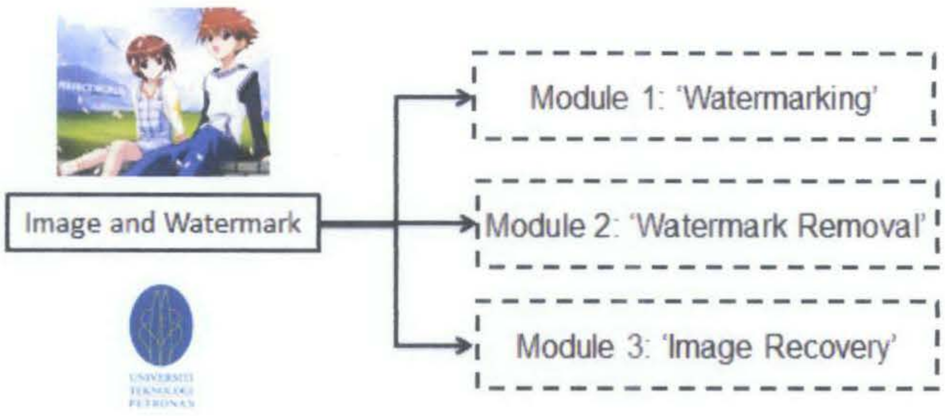


Figure 12: System Architecture

The first module is the watermarking module. This module functions to perform watermarking on the digital image using the memory watermarking technique. This module will read the image and watermark and finally display the watermarked image to the user.

The next module is the watermark removal module. It functions as to remove watermark from the watermark media. This module ensures the original image of the watermarked image can be obtained. The final module is the recovery module. The recovery module is to function as to check the integrity of the image loaded and if there exist any changes to the original images, it will restore the image back to its original value. Further explanation of how the modules and memory watermarking works is at section 3.3.2 System Design.

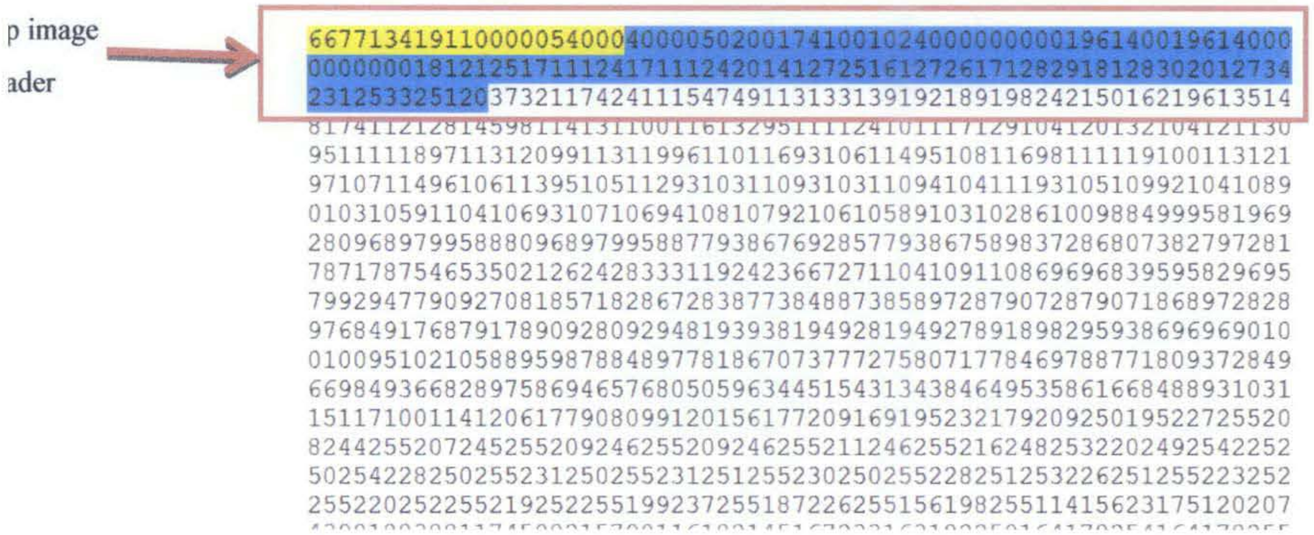
3.3.2 System Design

3.3.2.1Memory Watermarking

In the present, watermarking is done by embedding the watermark directly into the image by changing the bits of the images by using the embedding algorithm such as discrete wavelet transformation (DWT) and etc. The resulted watermarked image is save as a physical file such as jpeg, png or etc. which contains the watermarked image that being embedded into it.

In this system, the author is proposing memory watermarking technique to perform watermarking for digital images. Memory watermarking is a technique where the watermarking procedure is done in the memory and displayed to the users and it is different from the present technique used. There are no any physical watermarked image files that exist when using this technique as the image and the watermark are separated in different files. The watermarking is only done in the memory without affecting the original image.

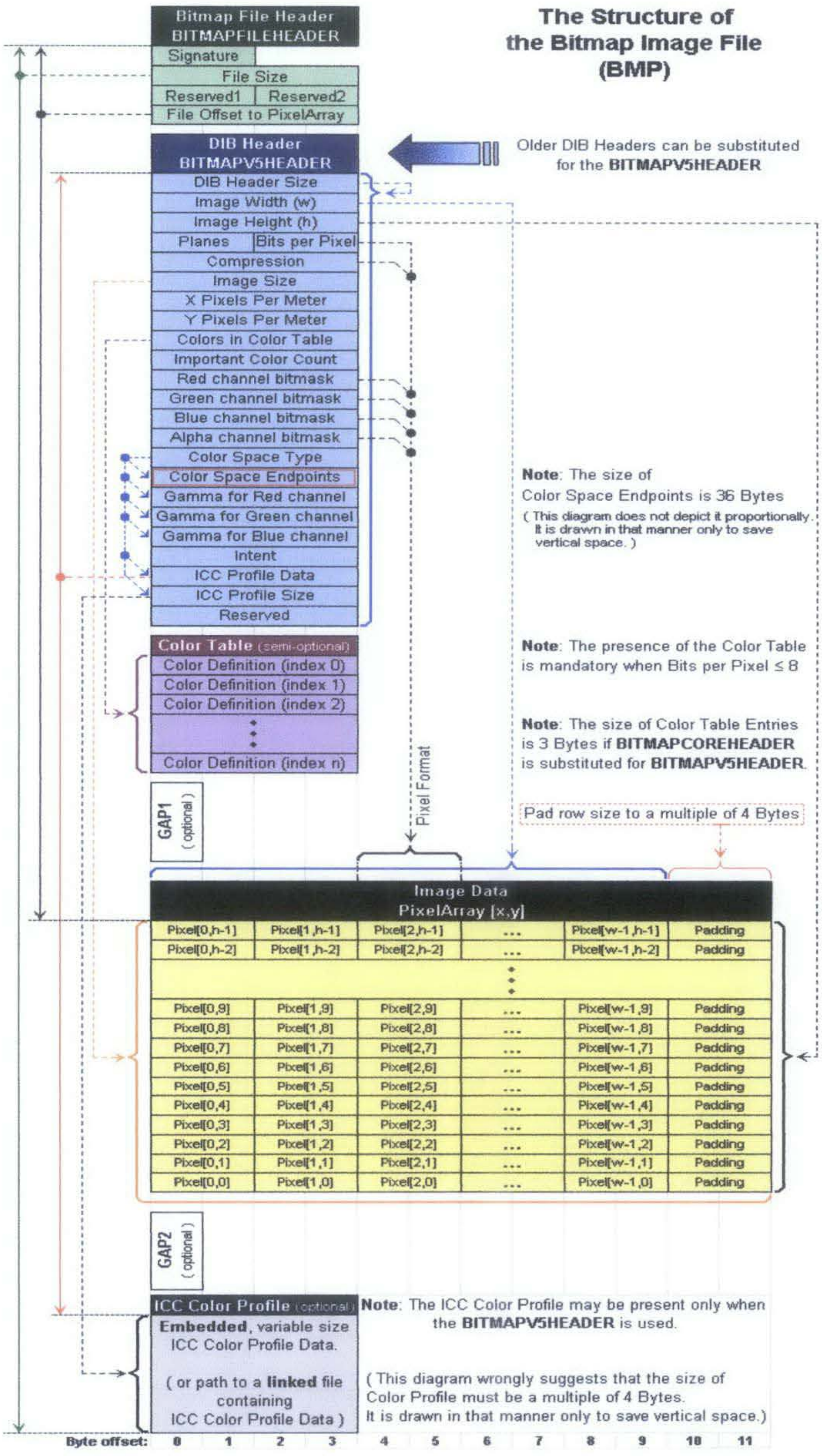
Firstly, the selected image read by a memory stream as sequence of bytes where it contains the image information such as the pixel value and etc. In this, we are taking the example of a bitmap file.



```
667713419110000054000400005020017410010240000000000196140019614000
000000018121251711124171112420141272516127261712829181283020127 34
231253325120373211742411154749113133139192189198242150162196135 14
81741121281459811413110011613295111124101117129104120132104121130
951111189711312099113119961101169310611495108116981111191001131 21
971071149610611395105112931031109310311094104111931051099210410 89
010310591104106931071069410810792106105891031028610098849995819 69
280968979958880968979958877938676928577938675898372868073827972 81
787178754653502126242833311924236672711041091108696968395958296 95
799294779092708185718286728387738488738589728790728790718689728 28
976849176879178909280929481939381949281949278918982959386969690 10
010095102105889598788489778186707377727580717784697887718093728 49
669849366828975869465768050596344515431343846495358616684889310 31
151117100114120617790809912015617720916919523217920925019522725 520
824425520724525520924625520924625521124625521624825322024925422 52
502542282502552312502552312512552302502552282512532262512552232 52
255220252255219252255199237255187226255156198255114156231751202 07
```
Figure 13: Bytes values of a bitmap image file.

From the image bytes value above, the one that highlighted in yellow is the bitmap file header. There is magic number which it represents the file format of the image. In bitmap it was unsigned integer 66 and 77 which represents the value BM. The value 134, 19, 11 and 0 represents the size of the bitmap file while the next 0, 0, 0 and 0 are unused. The final value of 54, 0, 0 and 0 represents the offset where the pixel array can be found.

The values highlighted in blue are the Device Independent Header (DIB). After the DIB, the values are the image pixels array containing the colour value of each pixel. The value here represented are in bytes which one (1) byte equals to eight (8) bits and the bytes values start from 0 to 255. The bitmap file structure is represented as in the image below.



The Structure of the Bitmap Image File (BMP)
Bitmap File Header
BITMAPFILEHEADER
Signature
File Size
Reserved1
Reserved2
File Offset to PixelArray
DIB Header
BITMAPV5HEADER
Older DIB Headers can be substituted for the BITMAPV5HEADER
DIB Header Size
Image Width (w)
Image Height (h)
Planes
Bits per Pixel
Compression
Image Size
X Pixels Per Meter
Y Pixels Per Meter
Colors in Color Table
Important Color Count
Red channel bitmask
Green channel bitmask
Blue channel bitmask
Alpha channel bitmask
Color Space Type
Color Space Endpoints
Gamma for Red channel
Gamma for Green channel
Gamma for Blue channel
Intent
ICC Profile Data
ICC Profile Size
Reserved
Note: The size of Color Space Endpoints is 36 Bytes
(This diagram does not depict it proportionally. It is drawn in that manner only to save vertical space.)
Color Table (semi-optional)
Color Definition (index 0)
Color Definition (index 1)
Color Definition (index 2)
Color Definition (index n)
Note: The presence of the Color Table is mandatory when Bits per Pixel ≤ 8
Note: The size of Color Table Entries is 3 Bytes if BITMAPCOREHEADER is substituted for BITMAPV5HEADER.
Pixel Format
GAP1 (optional)
Pad row size to a multiple of 4 Bytes
Image Data
PixelArray [x,y]
Pixel[0,h-1] Pixel[1,h-1] Pixel[2,h-1] ... Pixel[w-1,h-1] Padding
Pixel[0,h-2] Pixel[1,h-2] Pixel[2,h-2] ... Pixel[w-1,h-2] Padding
Pixel[0,9] Pixel[1,9] Pixel[2,9] ... Pixel[w-1,9] Padding
Pixel[0,8] Pixel[1,8] Pixel[2,8] ... Pixel[w-1,8] Padding
Pixel[0,7] Pixel[1,7] Pixel[2,7] ... Pixel[w-1,7] Padding
Pixel[0,6] Pixel[1,6] Pixel[2,6] ... Pixel[w-1,6] Padding
Pixel[0,5] Pixel[1,5] Pixel[2,5] ... Pixel[w-1,5] Padding
Pixel[0,4] Pixel[1,4] Pixel[2,4] ... Pixel[w-1,4] Padding
Pixel[0,3] Pixel[1,3] Pixel[2,3] ... Pixel[w-1,3] Padding
Pixel[0,2] Pixel[1,2] Pixel[2,2] ... Pixel[w-1,2] Padding
Pixel[0,1] Pixel[1,1] Pixel[2,1] ... Pixel[w-1,1] Padding
Pixel[0,0] Pixel[1,0] Pixel[2,0] ... Pixel[w-1,0] Padding
GAP2 (optional)
ICC Color Profile (optional)
Embedded, variable size ICC Color Profile Data.
(or path to a linked file containing ICC Color Profile Data)
Note: The ICC Color Profile may be present only when the BITMAPV5HEADER is used.
(This diagram wrongly suggests that the size of Color Profile must be a multiple of 4 Bytes. It is drawn in that manner only to save vertical space.)
Byte offset: 0 1 2 3 4 5 6 7 8 9 10 11

Figure 14: Bitmap file structure
(Source: http://en.wikipedia.org/wiki/BMP_file_format)

The value 37, 32, 117, 42 are the bytes value of the first pixel in the format of RGBA (red, green, blue, alpha). The array of the image pixels will build up the entire image where each pixel is represented in RGBA values. In this example, all the values are in bytes which 0xFF in bytes are 255.

Next in the process, the watermark image is also transformed into array of bytes in the memory stream. Using these bytes value, the images are built in the memory as a bitmap image object. Then, it is a graphic object was created out of this bitmap object using the Graphic Developer Interface (GDI+). Then, the watermark image is drawn to the graphic drawing surface of the graphic object with the position and the watermark opacity specified. Then, the graphic object that consist of the bitmap image and watermark are displayed to the user as watermarked image.

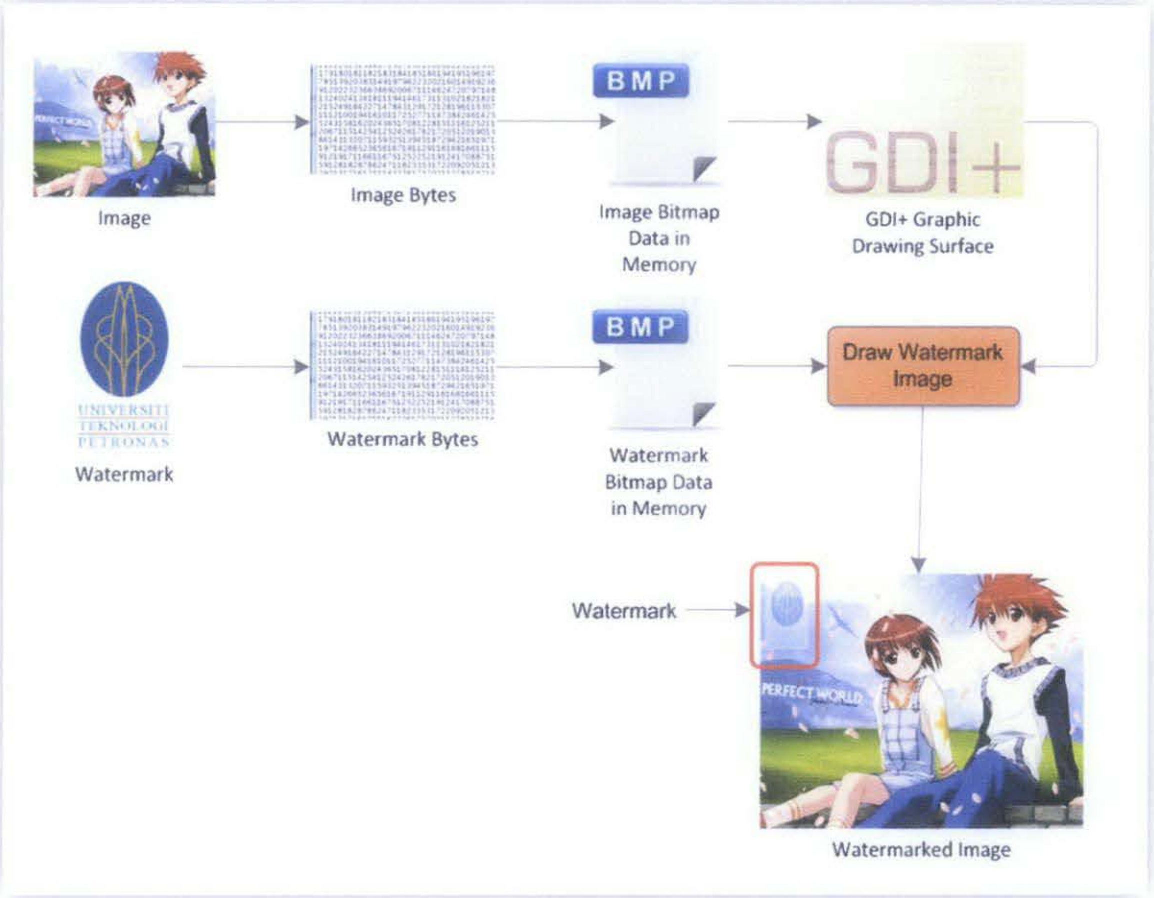


Image
Image Bytes
BMP
Image Bitmap Data in Memory
GDI+
GDI+ Graphic Drawing Surface
UNIVERSITI TEKNOLOGI PETRONAS
Watermark
Watermark Bytes
BMP
Watermark Bitmap Data in Memory
Draw Watermark Image
Watermark
PERFECT WORLD
Watermarked Image

Figure 15: How memory watermarking works.

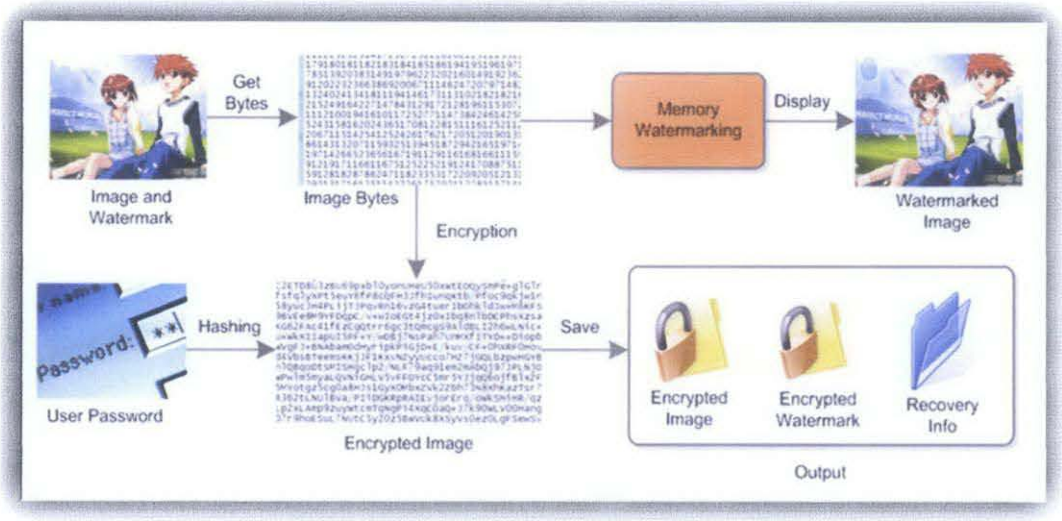


Figure 16: Graphical summary of the entire system.

3.3.2.2Saving Watermarked Image

When saving the watermark image, the users are required to enter the ownership information as well a password to protect the watermark from being removed. The user entered password will be hashed using SHA-512 which is known as Secure Hashing Algorithm. In the hashing algorithm, a strong pseudo-random number generator that uses the CryptAPI is generated as a salt to the hashing. This can improve the security of the password and reverse engineering of the password can prevent.

Using the hash of the password and the system generated salt, these values are hashed again and the final hash value of this are used to encrypt the images which are then converted to string from the bytes value. For the watermark, the same process also occurs where it is also encrypted with the final hash value of the salt and the hash. A header containing the password hash and salt are appended to the image and a file is generated. Another header containing the watermark information and the watermark ownership are appended to the encrypted watermark file and saved as another different file. Recovery information that contains both the watermark values and the images are stored in a different file encrypted.

All the three (3) files are then zipped together as one single file and saved with Watermarked Image (wmim) extension. The idea is adapted from docx format where a docx file contains various xml and cabinets files.

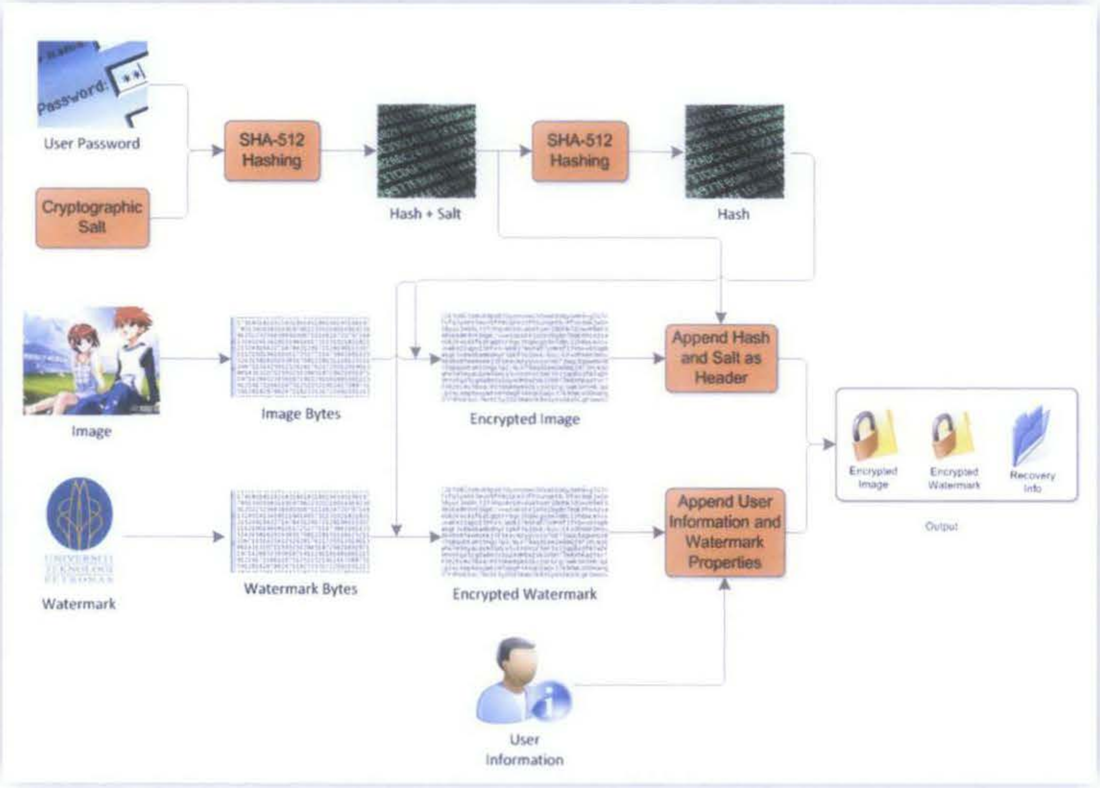
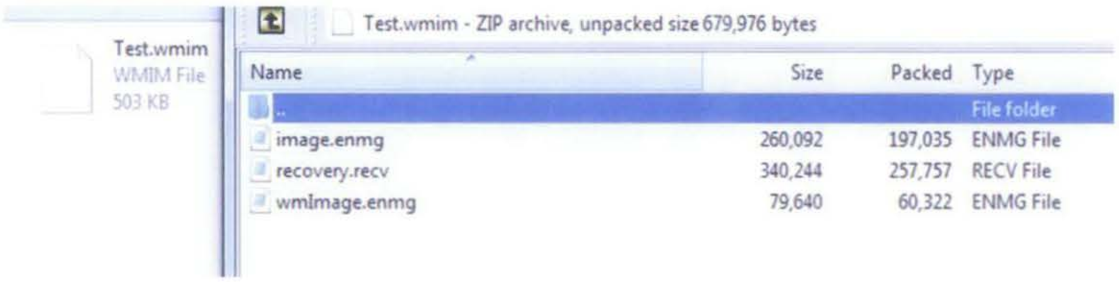


Figure 17: How watermark image is saved.


Test.wmim
WMIM File
503 KB

Test.wmim - ZIP archive, unpacked size 679,976 bytes

Name	Size	Packed	Type
..			File folder
image.enmg	260,092	197,035	ENMG File
recovery.recv	340,244	257,757	RECV File
wmImage.enmg	79,640	60,322	ENMG File

Figure 18: Output of the watermarking.

3.3.2.3 Watermark Removal

In the watermark removal process, the user needs to enter the password that they input during the creation of the watermark image file. The password entered by the user will be hashed with the salt in encrypted image file header and the resultant hash will be compared to the original hash. Only if the hash value matched, then the user can recover the original image. As the watermark is not directly embedded into the image itself, the originality of the image is maintained and user can select the format output type they want. With this feature, the watermark is reversible where it can be removed.

3.3.2.4Image Recovery

In the image recovery, the image bytes values are converted to base64 string. This is also similar to the watermark image and the watermark information will also be included in the recovery file.

The integrity of the image and watermark are check by calculating the SHA-512 hash of the image loaded in the memory stream. If found the value are different, then the recovery process started where the image information are loaded into the memory and the watermarking process continues. At the same time, the file will be overwritten with new values so that the next time the watermarked image file is loaded, it able to load correct image file.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Watermarking of the Digital Image

The author had conducted testing on the different image format such as Joint Photographic Expert Group (jpeg), Portable Network Graphic (png), Bitmap (bmp) and etc. The watermark types also are different where it was tested for both image watermark and also text watermark.

It shows that the watermarking can be done and presented to the users without the needs of the physical watermarked image file. Compared to the current watermarking techniques which embeds the watermark directly into the image and resulted in a physical watermarked file, the memory watermarking technique is able to display a watermarked image while having both the image and watermark separated in different file.

In the system, the watermarking of the images can be done in the memory while both the image and the watermark are separated in different file. The watermarked image in the memory is displayed to the users.

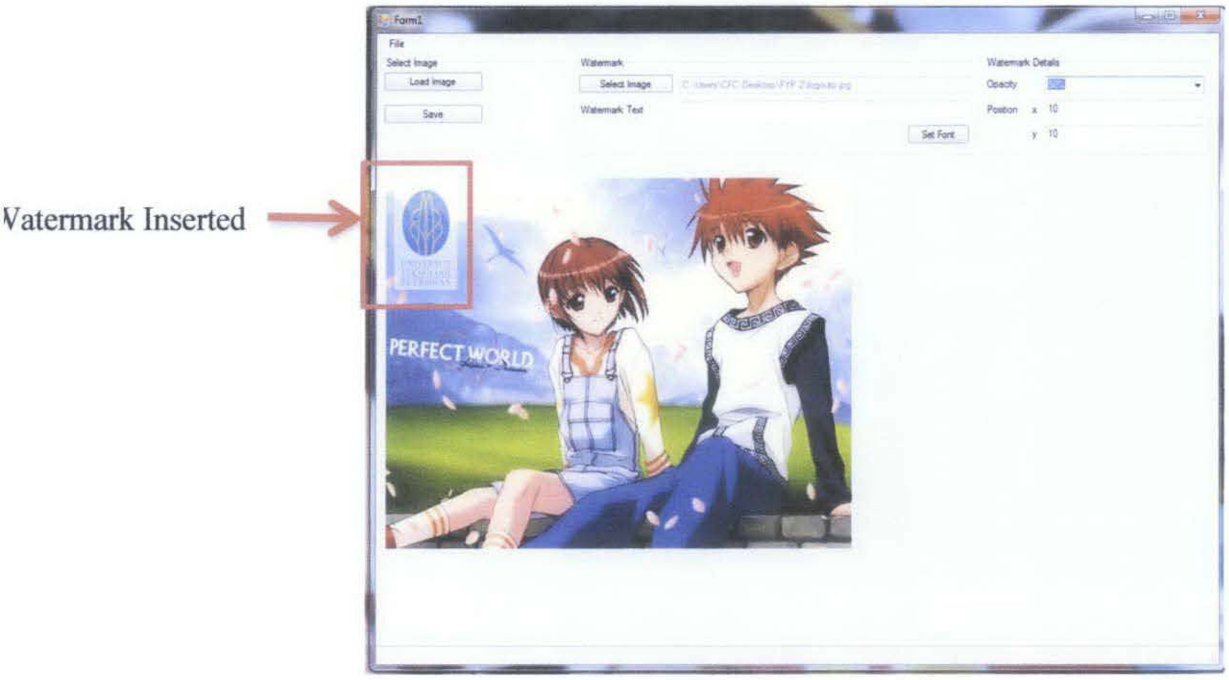


Figure 19: Watermarked Image

4.2 Restoration of Original Image

In the test conducted, the result of the values of the selected pixels of the original image and the restored image are recorded. The values of the pixels are compared using MatLab. By reading the image as matrices which represents the individual pixel value each, the original image and restored image are compared. The comparison process is done by subtracting the original image matrix with the restored image matrix. The result matrix of the subtraction is then sum up. If the value of the sum of the matrix is equivalent to zero (0), then both of the images are the same.

```
>> OriginalImage = imread('C:\Users\CFC\Desktop\Test\TestJpeg.jpg');
>> RestoredImage = imread('C:\Users\CFC\Desktop\Test\RestoredJpeg1.jpg');
>>
>> ResultSubtraction = OriginalImage - RestoredImage;
>>
>> SumResultColumns = sum(ResultSubtraction);
>>
>> SumAll = sum(SumResultColumns);
>>
```

Figure 20: Matlab code to compare the two image.

The sample test images are as attached in Appendix I. The results of the tests done on five sample images of different file type each are as follows:

File type	Image	Sum of Matrix Subtraction	Original Image = Restored Image
Jpeg	Jpeg Image A	Sum = 0	Yes
Jpeg	Jpeg Image B	Sum = 0	Yes
Jpeg	Jpeg Image C	Sum = 0	Yes
Jpeg	Jpeg Image D	Sum = 0	Yes
Jpeg	Jpeg Image E	Sum = 0	Yes
Bitmap	BMP Image A	Sum = 0	Yes
Bitmap	BMP Image B	Sum = 0	Yes
Bitmap	BMP Image C	Sum = 0	Yes
Bitmap	BMP Image D	Sum = 0	Yes
Bitmap	BMP Image E	Sum = 0	Yes
PNG	PNG Image A	Sum = 0	Yes
PNG	PNG Image B	Sum = 0	Yes
PNG	PNG Image C	Sum = 0	Yes
PNG	PNG Image D	Sum = 0	Yes
PNG	PNG Image E	Sum = 0	Yes

Table 1: Result of comparing image pixels between original image and restored image.

During the study, the author found that the restored image file could have different file size as well as the file hashes could be different between the original image and the restored image. After further analysis, the difference was found in the image header information. As for the image pixel information, there are no any differences between the original image and the restored image. The author found this problem where the image file headers are different between original image and the restored image for bitmap, jpeg and png files.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

This project highlights memory watermarking technique of embedding a watermark into a digital image which can be removed to get back the original image. It explores the possibility of performing the watermarking in the memory and not having the physical file that the watermark was embedded to it. The recovery scheme enables user to recover the image to restore it to the original condition if it was tampered with. The application of this can be applied to the medical images as well as software models and etc.

The final outcome of this project is able to achieve both the objectives which are as follows:

1. To introduce memory watermarking as a new way of reversible digital watermarking.
2. To achieve one hundred percent (100%) matching with original image after watermark removal.

There is vast improvement can be made to the system and this research to make it better. The author currently uses simple encryption and decryption which make it insecure. In the conducted study, the author does not give much attention on the security of the system. There is much room of improvement of this system in terms of the security of this system especially in the encryption of the file.

The file size of the watermarked image file can be improved and reduced especially in the bitmap data which is large. There should be a better way of compression method used to compress all the files so that the file size can be reduced but at the same time do not cause any information to be loss or missing. This is one of the future works that can be done to improve the system.

REFERENCES

i. Secure Hash Standard. (2008, October). *Federal Information Processiong Standards Publication.* United States of America: National Institute of Standards and Technology.

ii. Bandyopadhyay, S., Paul, T., & Raychoudhury, A. (2010). Invisible Digital Watermarking Through Encryption. *International Journal of Computer Applications, 4*(5), 18-20.

iii. Chen, M., & Sun, X. (2010). A Digital Image Watermarking of Self-recovery Base on the SPIHT Algorithm. *2nd International Conference on Signal Processing Systems. 2*, pp. 621-624 . Dalian: IEEE.

iv. Feng, J.-B., Chu, Y.-P., Lin, I.-C., & Tsai, C.-S. (2006). Reversible Watermarking: Current Status and Key Issues. *International Journal of Network Security, 2*(3), 161-171.

v. He, H.-J., Zhang, J.-S., & Tai, H.-M. (2009). Self-recovery Fragile Watermarking Using Block-Neighborhood Tampering Characterization. In S. Katzenbeisser, & A.-R. Sadeghi (Eds.), *Information Hiding* (Vol. 5806, pp. 132-145). Berlin: Springer Berlin / Heidelberg.

vi. Hsiang-Cheh, H., Jeng-Shyang, P., & Hsueh-Ming, H. (2004). Watermarking Based on Transform Domain. In P. Jeng-Shyang, H. Hsiang-Cheh, & J. Lakhmi C. (Eds.), *Intelligent Watermarking Techniques* (Vol. 7, pp. 147-163). Singapore: World Scientific Publishing Co. Pte. Ltd.

vii. Huang, H.-C., Pan, J.-S., & Hang, H.-M. (2004). An Introduction to Watermarking Techniques. In J.-S. Pan, H.-C. Huang, & L. Jain, *Intelligent Watermarking Techniques* (Vol. 7, pp. 3-40). Singapore: World Scientific Publishing.

viii. Jean-Luc, D., & Stephan, R. (2000). A Survey of Current Watermarking Techniques. In K. Stefan, & P. Fabien A.P. (Eds.), *Information Hiding : Techniques for Steganography and Digital Watermarking* (pp. 121-148). Boston: Artech House Books.

ix. Liew, S., & Jasni, M. (2011). Reversible Tamper Localization and Recovery Watermarking Scheme with Secure Hash. *European Journal of Scientific Research, 49*(2), 249-264.

x. Puhan, N., & Ho, A. (2005). Restoration in Secure Text Document Image Authentication Using Erasable Watermarks. *Computational Intelligence and Security, Part II* (pp. 661-668). Xi'an: Springer-Verlag Berlin Heidelberg.

xi. Shih, F. (2007). *Digital Watermarking and Steganography: Fundamentals and Techniques.* CRC Press.

xii. Yang, H., Li, C.-T., Sun, X., Yang, Y., & Cing. (2008). Removable Visible Image Watermarking Algorithm in the Discrete Cosine Transform Domain. *Journal of Electronic Imaging, 17*(3), 033008.

xiii. YongJie, W., Yao, Z., Jeng-Shyang, P., & ShaoWei, W. (2005). A Reversible Watermark Scheme Combined with Hash Function and Lossless Compression. In R. Khosla, R. Howlett, & L. Jain (Eds.), *Knowledge-Based Intelligent Information and Engineering Systems* (Vol. 3682, pp. 1168-1174). Heidelberg: Springer Berlin.

xiv. Zhou, X., Wang, S., Zhou, N., & Yu, J. (2010). An Erasable Watermarking Scheme for Exact Authentication of Chinese Word Documents. *3rd International Congress on Image and Signal Processing* (pp. 1156-1160). Yantai: IEEE.

APPENDICES

APPENDIX I

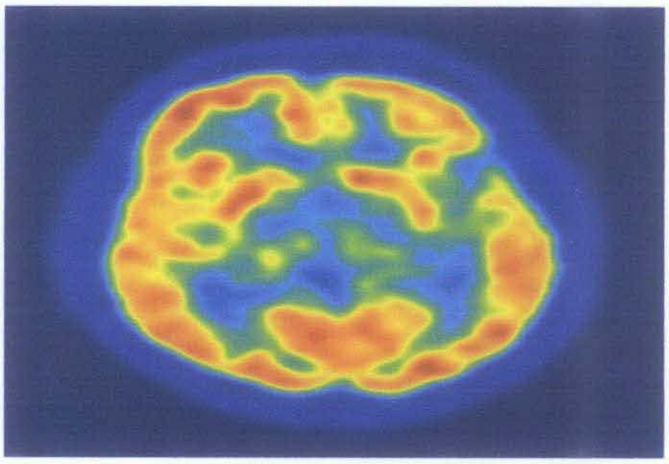
IMAGE USED IN TEST CASES

a) **Joint Photographic Expert Group (JPEG)**

i. **JPEG Image A**

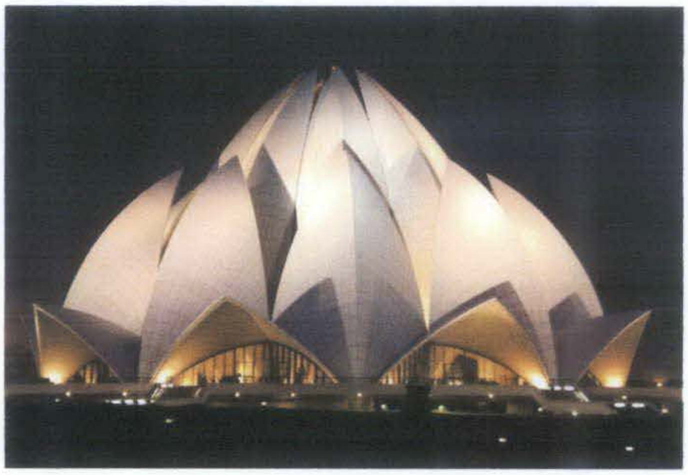
ii. **JPEG Image B**

iii. **JPEG Image C**



iv. **JPEG Image D**

v. **JPEG Image E**

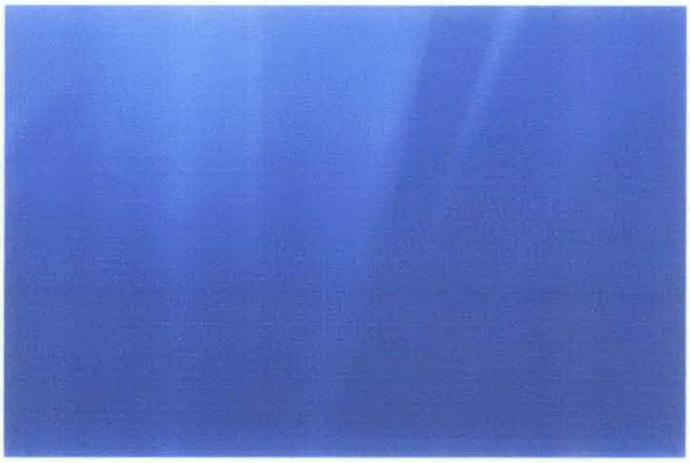


b) Portable Network Graphic (PNG)

i. PNG Image A

ii. PNG Image B

iii. PNG Image C



iv. **PNG Image D**

v. **PNG Image E**

c) Bitmap (BMP)

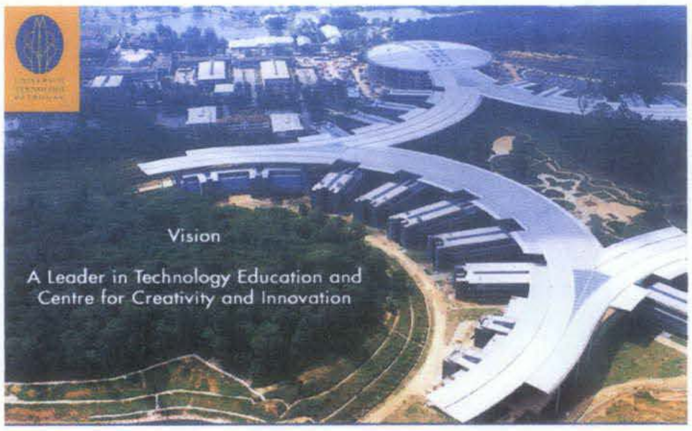
i. **BMP Image A**

ii. **BMP Image B**

iii. **BMP Image C**

iv. **BMP Image D**

v. **BMP Image E**


Vision
A Leader in Technology Education and
Centre for Creativity and Innovation

APPENDIX II

C# CODE: CLASS FORM1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Watermarking;

namespace FYP
{
  public partial class Form1 : Form
  {
    private string imageFilePath;
    private string watermarkImageFilePath;
    private Color textWatermarkColor;
    private Font textFont;
    private Watermark imgWatermarked;

    public Form1()
    {
      InitializeComponent();

      initSystem();
      initDropDownList();
      initTextbox();
    }

    public void initTextbox()
    {
      txtXpos.Text = "10";
      txtYpos.Text = "10";
    }

    public void initSystem()
    {
      btnRemoveWatermark.Visible = false;
      btnWatermarkInfo.Visible = false;
      textWatermarkColor = Color.Black;
      textFont = new Font("Times New Roman", 10);
    }

    public void initDropDownList()
    {
      string[] opacityList = new string[11] { "0%", "10%", "20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%",
"100%"};

      cbxOpacity.DataSource = opacityList;
    }

    public void loadImageFromFile(bool watermarked, string imageFilePath)
    {
      if (watermarked)
      {
        try
        {
          lblWatermark.Visible = true;
          lblWatermark.Text = "Watermarked Detected";
          lblWatermark.ForeColor = Color.Red;
```

```
            btnRemoveWatermark.Visible = true;
            btnWatermarkInfo.Visible = true;

            btnSelectWMImage.Enabled = false;
            btnSave.Enabled = false;
            btnFont.Enabled = false;
            txtWatermarkText.ReadOnly = true;
            txtXpos.ReadOnly = true;
            txtYpos.ReadOnly = true;
            cbxOpacity.Enabled = false;

            displayWatermarkedImage(imageFilePath);
        }
        catch
        {
            //exception
        }
    }
    else
    {
        try
        {
            lblWatermark.Text = "";
            lblWatermark.Visible = false;
            btnRemoveWatermark.Visible = false;
            btnWatermarkInfo.Visible = false;

            btnSelectWMImage.Enabled = true;
            btnSave.Enabled = true;
            btnFont.Enabled = true;
            txtWatermarkText.ReadOnly = false;
            txtXpos.ReadOnly = false;
            txtYpos.ReadOnly = false;
            cbxOpacity.Enabled = true;

            Image image = Image.FromFile(imageFilePath);
            pictureBox1.Image = image;

            pictureBox1.Size = image.Size;
        }
        catch
        {
            //exception
        }
    }
}

private void btnLoadImage_Click(object sender, EventArgs e)
{
    openFileDialog1.Title = "Open Image File";
    openFileDialog1.Filter = "Watermarked Image Files|*.wmim|JPEG Files|*.jpg" +
        "|Enhanced Windows MetaFile|*.emf" +
        "|Exchangeable Image File|*.exif" +
        "|Gif Files|*.gif|Bitmap Files|*.bmp" +
        "|PNG Files|*.png|TIFF Files|*.tif|Windows MetaFile|*.wmf";
    openFileDialog1.DefaultExt = "wmim";
    openFileDialog1.FilterIndex = 1;
    openFileDialog1.FileName = "";
    openFileDialog1.ShowDialog();

    // if the user did not select a file, return
    if (openFileDialog1.FileName == "")
        return;

    imageFilePath = openFileDialog1.FileName;
```

```
        if(IsWMIMFile(imageFilePath))
        {
            loadImageFromFile(true, imageFilePath);
        }
        else
        {
            loadImageFromFile(false, imageFilePath);
        }
    }

    static bool IsWMIMFile(string f)
    {
        return f != null && f.EndsWith(".wmim", StringComparison.Ordinal);
    }

    private void btnSave_Click(object sender, EventArgs e)
    {
        if (!string.IsNullOrEmpty(imageFilePath) && (!string.IsNullOrEmpty(watermarkImageFilePath) ||
!string.IsNullOrEmpty(txtWatermarkText.Text)))
        {
            //string password = Microsoft.VisualBasic.Interaction.InputBox("Please input your password?", "Input Password",
"");
            WatermarkDetails wmDetail = new WatermarkDetails();

            DialogResult result = wmDetail.ShowDialog();

            while(result != DialogResult.OK || wmDetail.Error)
            {
                if (result == DialogResult.Cancel)
                    break;
                else
                    result = wmDetail.ShowDialog();
            }

            if (!wmDetail.Error && (result == DialogResult.OK))
            {
                saveFileDialog1.Title = "Save Watermarked File";
                saveFileDialog1.FilterIndex = 1;
                saveFileDialog1.FileName = string.Empty;
                saveFileDialog1.Filter = "Watermarked Image Files|*.wmim";
                saveFileDialog1.DefaultExt = "wmim";

                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    if (saveFileDialog1.FileName == "")
                    {
                        MessageBox.Show("File name cannot be empty", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                        return;
                    }
                    else
                    {
                        int posY = getPosY();
                        int posX = getPosX();

                        float opacity = getOpacity();

                        //save watermark ownership details

                        if (!string.IsNullOrEmpty(watermarkImageFilePath))
                        {
                            new Watermark().SaveWatermarkedFile(saveFileDialog1.FileName, wmDetail.Password, imageFilePath,
watermarkImageFilePath, opacity, posY, posX, wmDetail.OwnerName, wmDetail.Organization, wmDetail.Email);
                        }
                        else
                        {
```

```
                    new Watermark().SaveWatermarkedFile(saveFileDialog1.FileName, wmDetail.Password, imageFilePath,
txtWatermarkText.Text, opacity, posY, posX, textWatermarkColor, textFont, wmDetail.OwnerName, wmDetail.Organization,
wmDetail.Email);
                  }
                }

                MessageBox.Show("File saved", "File Saved", MessageBoxButtons.OK, MessageBoxIcon.Information);
              }
            }
          }
          else
          {
            //display error
            if (string.IsNullOrEmpty(imageFilePath))
            {
              MessageBox.Show("There is no image selected to be watermarked", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            else if (string.IsNullOrEmpty(watermarkImageFilePath) && string.IsNullOrEmpty(txtWatermarkText.Text))
            {
              MessageBox.Show("There is no watermark image selected or text inserted", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
          }
        }

        private void btnSelectWMImage_Click(object sender, EventArgs e)
        {
          openFileDialog2.Title = "Open Image File";
          openFileDialog2.Filter = "JPEG Files|*.jpg" +
            "|Enhanced Windows MetaFile|*.emf" +
            "|Exchangeable Image File|*.exif" +
            "|Gif Files|*.gif|Bitmap Files|*.bmp" +
            "|PNG Files|*.png|TIFF Files|*.tif|Windows MetaFile|*.wmf";
          openFileDialog2.DefaultExt = "jpg";
          openFileDialog2.FilterIndex = 1;
          openFileDialog2.FileName = "";
          openFileDialog2.ShowDialog();

          // if the user did not select a file, return
          if (openFileDialog2.FileName == "")
            return;

          watermarkImageFilePath = openFileDialog2.FileName;
          txtWMImgPath.Text = watermarkImageFilePath;

          float opacity = getOpacity();
          int posX = getPosX();
          int posY = getPosY();
          displayWatermarkedImage(posX, posY, opacity);
        }

        private void cbxOpacity_SelectedIndexChanged(object sender, EventArgs e)
        {
          float opacity = getOpacity();
          int posX = getPosX();
          int posY = getPosY();

          displayWatermarkedImage(posX, posY, opacity);
        }

        private void txtXpos_TextChanged(object sender, EventArgs e)
        {
          float opacity = getOpacity();
          int posX = getPosX();
          int posY = getPosY();
```

```
    displayWatermarkedImage(posX, posY, opacity);
}

private void txtYpos_TextChanged(object sender, EventArgs e)
{
    float opacity = getOpacity();
    int posX = getPosX();
    int posY = getPosY();

    displayWatermarkedImage(posX, posY, opacity);
}

private float getOpacity()
{
    string opacityStr = cbxOpacity.SelectedValue.ToString();
    float opacity = 0;

    switch (opacityStr)
    {
        case "0%": opacity = 0;
            break;
        case "10%": opacity = 10;
            break;
        case "20%": opacity = 20;
            break;
        case "30%": opacity = 30;
            break;
        case "40%": opacity = 40;
            break;
        case "50%": opacity = 50;
            break;
        case "60%": opacity = 60;
            break;
        case "70%": opacity = 70;
            break;
        case "80%": opacity = 80;
            break;
        case "90%": opacity = 90;
            break;
        case "100%": opacity = 100;
            break;
    }

    return opacity;
}

private int getPosX()
{
    if (string.IsNullOrEmpty(txtXpos.Text))
        return 0;
    else
        return Convert.ToInt32(txtXpos.Text);
}

private int getPosY()
{
    if (string.IsNullOrEmpty(txtYpos.Text))
        return 0;
    else
        return Convert.ToInt32(txtYpos.Text);
}

private void displayWatermarkedImage(int posX, int posY, float opacity)
{
    if (!string.IsNullOrEmpty(watermarkImageFilePath) && !string.IsNullOrEmpty(imageFilePath))
```

```
            {
                Bitmap bmp = new Watermark().combineImage(opacity, Image.FromFile(watermarkImageFilePath),
Image.FromFile(imageFilePath), posX, posY);

                pictureBox1.Image = bmp;
                pictureBox1.Size = bmp.Size;
            }
            else
            {
                //error message
            }
        }

        private void displayWatermarkedImage(string filepath)
        {
            Watermark wm = new Watermark();

            Image im = wm.loadWatermarkedImage(filepath);
            pictureBox1.Image = im;
            pictureBox1.Size = im.Size;

            imgWatermarked = wm;
        }

        private void displayWatermarkedImage(int posX, int posY, float opacity, string text)
        {
            if (!string.IsNullOrEmpty(imageFilePath))
            {
                Bitmap bmp = new Watermark().combineText(opacity, Image.FromFile(imageFilePath), text, textFont,
textWatermarkColor, posX, posY);

                pictureBox1.Image = bmp;
                pictureBox1.Size = bmp.Size;
            }
        }

        private void btnFont_Click(object sender, EventArgs e)
        {
            if (fontDialog1.ShowDialog() != DialogResult.Cancel)
            {
                textFont = fontDialog1.Font;
                textWatermarkColor = fontDialog1.Color;

                string WatermarkText = txtWatermarkText.Text;

                float opacity = getOpacity();
                int posX = getPosX();
                int posY = getPosY();

                displayWatermarkedImage(posX, posY, opacity, WatermarkText);
            }
        }

        private void txtWatermarkText_TextChanged(object sender, EventArgs e)
        {
            string WatermarkText = txtWatermarkText.Text;

            float opacity = getOpacity();
            int posX = getPosX();
            int posY = getPosY();

            displayWatermarkedImage(posX, posY, opacity, WatermarkText);
        }

        private void btnRemoveWatermark_Click(object sender, EventArgs e)
        {
```

```
        string password = Microsoft.VisualBasic.Interaction.InputBox("Please input your password?", "Input Password", "");

        byte[] imageBytes = new Watermark().checkWatermarkPassword(password, imageFilePath);

        if (imageBytes == null)
        {
          MessageBox.Show("Incorrect Password", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
          return;
        }
        else
        {
          saveFileDialog2.Title = "Save Image File";
          saveFileDialog2.FilterIndex = 1;
          saveFileDialog2.FileName = string.Empty;
          saveFileDialog2.Filter = "JPEG Files|*.jpg" +
                "|Enhanced Windows MetaFile|*.emf" +
                "|Exchangeable Image File|*.exif" +
                "|Gif Files|*.gif|Bitmap Files|*.bmp" +
                "|PNG Files|*.png|TIFF Files|*.tiff|Windows MetaFile|*.wmf";
          saveFileDialog2.DefaultExt = "jpg";

          if (saveFileDialog2.ShowDialog() == DialogResult.OK)
          {
            if (saveFileDialog2.FileName == "")
            {
              MessageBox.Show("File name cannot be empty", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
              return;
            }
            else
            {
              new Watermark().SaveImage(saveFileDialog2.FileName, imageBytes, saveFileDialog2.FilterIndex);
              MessageBox.Show("File saved", "File Saved", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
          }
        }
      }

      private void btnWatermarkInfo_Click(object sender, EventArgs e)
      {
        string []owner = imgWatermarked.OwnerDetail;

        string displayOwner = string.Format("Owner's Name : {0}\nOrganization : {1}\nE-Mail : {2}\n", owner[0], owner[1],
owner[2]);

        MessageBox.Show(displayOwner, "Owner Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
      }

    }

}
```

APPENDIX III

C# CODE: CLASS WATERMARK DETAILS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace FYP
{
  public partial class WatermarkDetails : Form
  {
    private string password;
    private string ownerName;
    private string organization;
    private string email;
    private bool error;

    public WatermarkDetails()
    {
      InitializeComponent();
      password = string.Empty;
      ownerName = string.Empty;
      organization = string.Empty;
      email = string.Empty;
    }

    private void btnConfirm_Click(object sender, EventArgs e)
    {
      error = false;

      if (!string.IsNullOrEmpty(txtPassword.Text) && !string.IsNullOrEmpty(txtConfirmPassword.Text))
      {
        if(!txtPassword.Text.Equals(txtConfirmPassword.Text))
        {
          MessageBox.Show("Password and Confirm Password value is different", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
          return;
        }
      }

      string errorMsg = string.Empty;
      bool encounterError = false;

      if (string.IsNullOrEmpty(txtOwnerName.Text))
      {
        encounterError = true;
        errorMsg = string.Format(errorMsg + "Owner's Name is required.\n");
      }

      if (string.IsNullOrEmpty(txtOrganization.Text))
      {
        encounterError = true;
        errorMsg = string.Format(errorMsg + "Organization is required.\n");
      }

      if (string.IsNullOrEmpty(txteMail.Text))
    {
        encounterError = true;
        errorMsg = string.Format(errorMsg + "E-mail address is required.\n");
    }

    if (string.IsNullOrEmpty(txtPassword.Text))
    {
        encounterError = true;
        errorMsg = string.Format(errorMsg + "Password is required.\n");
    }

    if (string.IsNullOrEmpty(txtConfirmPassword.Text))
    {
        encounterError = true;
        errorMsg = string.Format(errorMsg + "Confirm Password is required.\n");
    }

    string emailExpression = @"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*";
    Regex emailRe = new Regex(emailExpression);

    if (!string.IsNullOrEmpty(txteMail.Text) && !emailRe.IsMatch(txteMail.Text))
    {
        encounterError = true;
        errorMsg = string.Format(errorMsg + "E-Mail entered format is wrong.\n");
    }

    if (encounterError)
    {
        MessageBox.Show(errorMsg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        error = true;
    }

    if (!error)
    {
        password = txtPassword.Text;
        ownerName = txtOwnerName.Text;
        organization = txtOrganization.Text;
        email = txteMail.Text;
        Close();
    }
}

private void btnCancel_Click(object sender, EventArgs e)
{
    Close();
}

public string Password
{
    get
    {
        return password;
    }
}

public string OwnerName
{
    get
    {
        return ownerName;
    }
}

public string Organization
{
    get
```

```
            {
                return organization;
            }
        }

        public string Email
        {
            get
            {
                return email;
            }
        }

        public bool Error
        {
            get
            {
                return error;
            }
        }

    }
}
```

APPENDIX IV

C# CODE: CLASS PROJECT HELPER

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ProjectHelper
{
  public class ConstantHelper
  {
    public class AppsSetting
    {
      public const string TEMP_FILE_PATH = @"C:\Temp\";
      public const string IMAGE = "image.enmg";
      public const string WATERMARK_IMAGE = "wmImage.enmg";
      public const string RECOVERY = "recovery.recv";
    }
  }
}
```

APPENDIX V

C# CODE: CLASS WATERMARKING

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using Security;
using Ionic.Zip;
using ProjectHelper;
using System.ComponentModel;
using System.Security.Cryptography;
using Recovery;

namespace Watermarking
{
  public class Watermark
  {
    private string[] ownerDetail;

    public Image loadWatermarkedImage(string filePath)
    {
      return loadWMImageFile(filePath);
    }

    public void SaveWatermarkedFile(string fileName, string password, string ImagePath, string WatermarkImagePath, float
opacity, int posY, int posX, string ownerName, string organization, string email)
    {
      string Hash;
      string Salt;

      Hashing _hashObj = new Hashing();
      EncryptionandDecryption _EncandDecObj1 = new EncryptionandDecryption();
      EncryptionandDecryption _EncandDecObj2 = new EncryptionandDecryption();
      EncryptionandDecryption _EncandDecObj3 = new EncryptionandDecryption();
      RecoveryScheme _RecoveryObj = new RecoveryScheme();

      _hashObj.GetHashAndSaltString(password, out Hash, out Salt);

      string imageString = convertImageToBase64String(ImagePath);
      string watermarkImageString = convertImageToBase64String(WatermarkImagePath);


      //image
      string encPass = _hashObj.GetHash(Hash, Salt);

      string encryptedImageString = _EncandDecObj1.EncryptString(imageString, encPass);

      //append header that contain the salt and hash value of the password
      string header = Hash + Salt;
      encryptedImageString = header + encryptedImageString;

      writeToFile(encryptedImageString, true, fileName, false);


      //Watermark
      string wmPass = _hashObj.GetHash(Salt, Hash);

      //owner
      string watermarkOnwerString = string.Format("{0};{1};{2};", ownerName, organization, email);
```

```
            string wmOnwerBase64 = Convert.ToBase64String(Encoding.ASCII.GetBytes(watermarkOnwerString));
            string encryptedOwner = _EncandDecObj2.EncryptString(wmOnwerBase64, wmPass);

            string encryptedWatermarkString = _EncandDecObj3.EncryptString(watermarkImageString, wmPass);

            //watermark header contains the watermark information and position + opacity
            string wmHeader = string.Format("{0};{1};{2};{3};", opacity, posX, posY, true);

            string wmHeaderBase64 = Convert.ToBase64String(Encoding.ASCII.GetBytes(wmHeader));
            encryptedWatermarkString = string.Format("{0}//;//{1}//;//{2}", wmHeader, encryptedOwner,
encryptedWatermarkString);

            writeToFile(encryptedWatermarkString, false, fileName, false);

            string recoveryInfo = _RecoveryObj.createRecoveryFile(imageString, watermarkImageString, Hash, Salt,
wmOnwerBase64, wmHeaderBase64, true);

            writeToFile(recoveryInfo, false, fileName, true);

        }

        public void SaveWatermarkedFile(string fileName, string password, string ImagePath, string WatermarkText, float
opacity, int posY, int posX, Color myColor, Font myFont, string ownerName, string organization, string email)
        {
            string Hash;
            string Salt;

            Hashing _hashObj = new Hashing();
            EncryptionandDecryption _EncandDecObj1 = new EncryptionandDecryption();
            EncryptionandDecryption _EncandDecObj2 = new EncryptionandDecryption();
            EncryptionandDecryption _EncandDecObj3 = new EncryptionandDecryption();
            RecoveryScheme _RecoveryObj = new RecoveryScheme();

            _hashObj.GetHashAndSaltString(password, out Hash, out Salt);

            string imageString = convertImageToBase64String(ImagePath);

            //image
            string encPass = _hashObj.GetHash(Hash, Salt);

            string encryptedImageString = _EncandDecObj1.EncryptString(imageString, encPass);

            //append header that contain the salt and hash value of the password
            string header = Hash + Salt;
            encryptedImageString = header + encryptedImageString;

            writeToFile(encryptedImageString, true, fileName, false);


            //Watermark
            string wmPass = _hashObj.GetHash(Salt, Hash);

            TypeConverter tcFont = TypeDescriptor.GetConverter(typeof(Font));
            string fontString = tcFont.ConvertToString(myFont);

            TypeConverter tcColor = TypeDescriptor.GetConverter(typeof(Color));
            string colorString = tcColor.ConvertToString(myColor);

            string watermarkString = string.Format("{0};{1};{2};", WatermarkText, colorString, fontString);

            string watermarkOnwerString = string.Format("{0};{1};{2};", ownerName, organization, email);
            string wmOnwerBase64 = Convert.ToBase64String(Encoding.ASCII.GetBytes(watermarkOnwerString));
            string encryptedOwner = _EncandDecObj2.EncryptString(wmOnwerBase64 ,wmPass);

            string encryptedWatermarkString = _EncandDecObj3.EncryptString(watermarkString, wmPass);
        //watermark header contains the watermark information and position + opacity
        string wmHeader = string.Format("{0};{1};{2};{3};", opacity, posX, posY, false);

        string wmHeaderBase64 = Convert.ToBase64String(Encoding.ASCII.GetBytes(wmHeader));
        encryptedWatermarkString = string.Format("{0}//;//{1}//;//{2}", wmHeader, encryptedOwner,
encryptedWatermarkString);

        writeToFile(encryptedWatermarkString, false, fileName, false);

        string recoveryInfo = _RecoveryObj.createRecoveryFile(imageString, watermarkString, Hash, Salt, wmOnwerBase64,
wmHeaderBase64, false);

        writeToFile(recoveryInfo, false, fileName, true);
    }

    private void writeToFile(string content, bool isImageFile, string fileName, bool isRecovery)
    {
        string tempFileName;
        string tempImagePath = string.Format("{0}{1}", ConstantHelper.AppsSetting.TEMP_FILE_PATH,
ConstantHelper.AppsSetting.IMAGE);
        string tempWatermarkPath = string.Format("{0}{1}", ConstantHelper.AppsSetting.TEMP_FILE_PATH,
ConstantHelper.AppsSetting.WATERMARK_IMAGE);
        string tempRecoveryPath = string.Format("{0}{1}", ConstantHelper.AppsSetting.TEMP_FILE_PATH,
ConstantHelper.AppsSetting.RECOVERY);

        if (isImageFile)
            tempFileName = tempImagePath;
        else if(isRecovery)
            tempFileName = tempRecoveryPath;
        else
            tempFileName = tempWatermarkPath;

        using (StreamWriter sw = new StreamWriter(tempFileName))
        {
            sw.Write(content);
        }

        if (isRecovery)
        {
            using (ZipFile zip = new ZipFile())
            {
                String[] filenames = { tempImagePath, tempWatermarkPath, tempRecoveryPath };
                zip.CompressionLevel = Ionic.Zlib.CompressionLevel.BestCompression;
                zip.AddFiles(filenames, "");
                zip.Save(string.Format("{0}", fileName));
            }

            File.Delete(tempImagePath);
            File.Delete(tempWatermarkPath);
            File.Delete(tempRecoveryPath);
        }


    }

    private string convertImageToBase64String(string ImagePath)
    {
        Image oriImage;

        oriImage = Image.FromFile(ImagePath);
        byte[] bytestream = imageToByteArray(oriImage);

        string imageString = Convert.ToBase64String(bytestream);

        return imageString;
    }
    private byte[] imageToByteArray(Image imageIn)
    {
      MemoryStream ms = new MemoryStream();
      ImageFormat format = imageIn.RawFormat;
      imageIn.Save(ms, format);
      //imageIn.Save(ms, ImageFormat.Bmp);
      return ms.ToArray();
    }

    private string readFromFile(string filepath)
    {
      string str;

      using (StreamReader sr = new StreamReader(filepath))
      {
        str = sr.ReadToEnd();
      }

      return str;
    }

    private Image byteArrayToImage(byte[] byteArrayIn)
    {
      MemoryStream ms = new MemoryStream(byteArrayIn);
      Image returnImage = Image.FromStream(ms);
      return returnImage;
    }

    public Bitmap combineImage(float opacityvalue, Image watermarkImage, Image oriImage, int xPosOfWm, int
yPosOfWm)
    {
      opacityvalue = opacityvalue / 100;

      Bitmap bmPhoto = new Bitmap(oriImage);
      Graphics grPhoto = Graphics.FromImage(bmPhoto);

      int wmimgHeight = watermarkImage.Height;
      int wmimgWidth = watermarkImage.Width;

      ImageAttributes imageAttributes = new ImageAttributes();

      ColorMatrix colormatrix = new ColorMatrix();
      colormatrix.Matrix33 = opacityvalue;

      imageAttributes.SetColorMatrix(colormatrix, ColorMatrixFlag.Default, ColorAdjustType.Bitmap);

      grPhoto.DrawImage(watermarkImage,
        new Rectangle(xPosOfWm, yPosOfWm, wmimgWidth, wmimgHeight), //Set the detination Position
        0,            // x-coordinate of the portion of the source image to draw.
        0,            // y-coordinate of the portion of the source image to draw.
        wmimgWidth,        // Watermark Width
        wmimgHeight,                 // Watermark Height
        GraphicsUnit.Pixel, // Unit of measurment
        imageAttributes); //ImageAttributes Object

      return bmPhoto;
    }

    public Bitmap combineText(float opacityvalue, Image oriImage, string textWatermark, Font myFont, Color
myWatermarkColor, int posX, int posY)
    {
      int opacity = Convert.ToInt32(opacityvalue / 100 * 255);

      Bitmap bmPhoto = new Bitmap(oriImage);
      Graphics grPhoto = Graphics.FromImage(bmPhoto);
```

```
        // Create a solid brush to write the watermark text on the image
        Brush myBrush = new SolidBrush(Color.FromArgb(opacity, myWatermarkColor));

        // Calculate the size of the text
        SizeF sz = grPhoto.MeasureString(textWatermark, myFont);

        // draw the water mark text
        grPhoto.DrawString(textWatermark, myFont, myBrush, new Point(posX, posY));

        return bmPhoto;

    }

    private Image loadWMImageFile(string filepath)
    {
        var wmMemoryStream = new MemoryStream();
        var imageMemoryStream = new MemoryStream();

        using (ZipFile zip = ZipFile.Read(filepath))
        {
            ZipEntry imentry = zip[ConstantHelper.AppsSetting.IMAGE];
            imentry.Extract(imageMemoryStream);

            ZipEntry wmentry = zip[ConstantHelper.AppsSetting.WATERMARK_IMAGE];
            wmentry.Extract(wmMemoryStream);
        }

        EncryptionandDecryption _EncandDecObj1 = new EncryptionandDecryption();
        EncryptionandDecryption _EncandDecObj2 = new EncryptionandDecryption();
        EncryptionandDecryption _EncandDecObj3 = new EncryptionandDecryption();
        Hashing _hashObj = new Hashing();
        RecoveryScheme _recoveryObj = new RecoveryScheme();

        //image
        string imageFileContent = Encoding.ASCII.GetString(imageMemoryStream.ToArray());

        string Hash = imageFileContent.Substring(0,88) ;
        string Salt = imageFileContent.Substring(88,12) ;

        string encryptedImage = imageFileContent.Substring(100);

        string encPass = _hashObj.GetHash(Hash, Salt);

        string decryptedImageString = _EncandDecObj1.DecryptString(encryptedImage, encPass);
        byte[] imageBytes = Convert.FromBase64String(decryptedImageString);


        if (!_recoveryObj.compareHash(filepath, imageBytes, true))
        {
            //perform recovery
            string recovery = _recoveryObj.performRecovery(filepath, ConstantHelper.AppsSetting.IMAGE, true);

            imageBytes = Convert.FromBase64String(recovery);
        }

        //watermark
        string [] separator = {"//;//"};
        string wmFileContent = Encoding.ASCII.GetString(wmMemoryStream.ToArray());
        string[] wmContent = wmFileContent.Split(separator, 3, StringSplitOptions.None);

        string wmInfo = wmContent[0];
        string encryptedOnwer = wmContent[1];
        string encryptedWMImage = wmContent[2];

        //wm info
        string[] info = wmInfo.Split(';');

        float opacityvalue = float.Parse(info[0]);
        int posX = Convert.ToInt32(info[1]);
        int posY = Convert.ToInt32(info[2]);
        bool isImage = Convert.ToBoolean(info[3]);

        //watermark password
        string encWMPass = _hashObj.GetHash(Salt, Hash);

        //watermark owner details
        string decryptedOwner64String = _EncandDecObj2.DecryptString(encryptedOnwer, encWMPass);
        byte[] byteOnwer = Convert.FromBase64String(decryptedOwner64String);
        string wmOwner = Encoding.ASCII.GetString(byteOnwer);
        ownerDetail = wmOwner.Split(';');

        string decryptedWMString = _EncandDecObj3.DecryptString(encryptedWMImage, encWMPass);
        Bitmap bmp = null;
        Image image = byteArrayToImage(imageBytes);

        if (isImage)
        {
          byte[] wmImageBytes = Convert.FromBase64String(decryptedWMString);

          if (!_recoveryObj.compareHash(filepath, wmImageBytes, false))
          {
            string recovery = _recoveryObj.performRecovery(filepath, ConstantHelper.AppsSetting.WATERMARK_IMAGE,
false);

            wmImageBytes = Convert.FromBase64String(recovery);
          }

          Image wmImage = byteArrayToImage(wmImageBytes);

          bmp = combineImage(opacityvalue, wmImage, image, posX, posY);
        }
        else
        {
          byte[] wmBytes = Encoding.ASCII.GetBytes(decryptedWMString);
          if (!_recoveryObj.compareHash(filepath, wmBytes, false))
          {
            //perform recovery
            string recovery = _recoveryObj.performRecovery(filepath, ConstantHelper.AppsSetting.WATERMARK_IMAGE,
false);

            decryptedWMString = recovery;
          }

          string[] textInfo = decryptedWMString.Split(';');

          TypeConverter tcFont = TypeDescriptor.GetConverter(typeof(Font));
          Font myFont = (Font)tcFont.ConvertFromString(textInfo[2]);

          TypeConverter tcColor = TypeDescriptor.GetConverter(typeof(Color));
          Color myColor = (Color)tcColor.ConvertFromString(textInfo[1]);

          bmp = combineText(opacityvalue, image, textInfo[0], myFont, myColor, posX, posY);
        }

        return bmp;
      }

      public byte[] checkWatermarkPassword(string password, string filepath)
      {
        var imageMemoryStream = new MemoryStream();
```

```
    using (ZipFile zip = ZipFile.Read(filepath))
    {
      ZipEntry imentry = zip[ConstantHelper.AppsSetting.IMAGE];
      imentry.Extract(imageMemoryStream);
    }

    string imageFileContent = Encoding.ASCII.GetString(imageMemoryStream.ToArray());

    string Hash = imageFileContent.Substring(0, 88);
    string Salt = imageFileContent.Substring(88, 12);

    Hashing _hashObj = new Hashing();
    string hashValue = _hashObj.GetHash(password, Salt);

    if (Hash.Equals(hashValue))
    {
      string encryptedImage = imageFileContent.Substring(100);

      return removeWatermark(Hash, Salt, encryptedImage);
    }
    else
    {
      return null;
    }
  }

  private byte[] removeWatermark(string Hash, string Salt, string encryptedImage)
  {
    EncryptionandDecryption _EncandDecObj1 = new EncryptionandDecryption();

    Hashing _hashObj = new Hashing();
    string encPass = _hashObj.GetHash(Hash, Salt);

    string decryptedImageString = _EncandDecObj1.DecryptString(encryptedImage, encPass);
    byte[] imageBytes = Convert.FromBase64String(decryptedImageString);

    return imageBytes;
  }

  public void SaveImage(string path, byte[] imageByte, int filterIndex)
  {
    MemoryStream ms = new MemoryStream(imageByte);
    Image image = Image.FromStream(ms);

    switch (filterIndex)
    {
      case 1:
        //path = string.Format("{0}.jpg", path);
        image.Save(path, ImageFormat.Jpeg);
        break;

      case 2:
        //path = string.Format("{0}.emf", path);
        image.Save(path, ImageFormat.Emf);
        break;

      case 3:
        //path = string.Format("{0}.exif", path);
        image.Save(path, ImageFormat.Exif);
        break;

      case 4:
        //path = string.Format("{0}.gif", path);
        image.Save(path, ImageFormat.Gif);
        break;
```

```
            case 5:
              //path = string.Format("{0}.bmp", path);
              image.Save(path, ImageFormat.Bmp);
              break;

            case 6:
              //path = string.Format("{0}.png", path);
              image.Save(path, ImageFormat.Png);
              break;

            case 7:
              //path = string.Format("{0}.tiff", path);
              image.Save(path, ImageFormat.Tiff);
              break;

            case 8:
              //path = string.Format("{0}.wmf", path);
              image.Save(path, ImageFormat.Wmf);
              break;

          }
        }

        public string[] OwnerDetail
        {
          get
          {
            return ownerDetail;
          }
        }
    }
}
```

APPENDIX IX

C# CODE: CLASS HASHING

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;

namespace Security
{
  public class Hashing
  {
    private HashAlgorithm HashProvider;
    private int SaltLength;

    public Hashing(HashAlgorithm HashProvider, int SaltLength)
    {
      this.HashProvider = HashProvider;
      this.SaltLength = SaltLength;
    }

    public Hashing()
    {
      HashProvider = new SHA512Managed();
      SaltLength = 8;
    }

    private byte[] ComputeHash(byte[] Data, byte[] Salt)
    {
      // Allocate memory to store both the Data and Salt together
      byte[] DataAndSalt = new byte[Data.Length + SaltLength];

      // Copy both the data and salt into the new array
      Array.Copy(Data, DataAndSalt, Data.Length);
      Array.Copy(Salt, 0, DataAndSalt, Data.Length, SaltLength);

      // Calculate the hash
      // Compute hash value of our plain text with appended salt.
      return HashProvider.ComputeHash(DataAndSalt);
    }

    public void GetHashAndSalt(byte[] Data, out byte[] Hash, out byte[] Salt)
    {
      // Allocate memory for the salt
      Salt = new byte[SaltLength];

      // Strong runtime pseudo-random number generator, on Windows uses CryptAPI
      // on Unix /dev/urandom
      RNGCryptoServiceProvider random = new RNGCryptoServiceProvider();

      // Create a random salt
      random.GetNonZeroBytes(Salt);

      // Compute hash value of our data with the salt.
      Hash = ComputeHash(Data, Salt);
    }

    public void GetHashAndSaltString(string Data, out string Hash, out string Salt)
    {
      byte[] HashOut;
      byte[] SaltOut;
      // Obtain the Hash and Salt for the given string
      GetHashAndSalt(Encoding.UTF8.GetBytes(Data), out HashOut, out SaltOut);

      // Transform the byte[] to Base-64 encoded strings
      Hash = Convert.ToBase64String(HashOut);
      Salt = Convert.ToBase64String(SaltOut);
    }

    public string GetHash(string Data, string Salt)
    {
      byte[] HashOut;

      HashOut = ComputeHash(Encoding.UTF8.GetBytes(Data), Convert.FromBase64String(Salt));

      return Convert.ToBase64String(HashOut);
    }
  }
}
```

APPENDIX VII

C# CODE: CLASS ENCRYPTION AND DECRYPTION

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;
using System.IO;

namespace Security
{
  public class EncryptionandDecryption
  {

    public string EncryptString(string InputText, string HashValue)
    {
      // "hash value" string variable the key(your secret key)
      // "InputText" string variable is the text to be encrypted.
      // We are now going to create an instance of the
      // Rihndael class.
      RijndaelManaged RijndaelCipher = new RijndaelManaged();

      byte[] PlainText = Encoding.Unicode.GetBytes(InputText);

      // We are using Salt to make it harder to guess our key
      // using a dictionary attack.
      byte[] Salt = Encoding.ASCII.GetBytes(HashValue.Length.ToString());

      // The (Secret Key) will be generated from the specified
      // HashValue and Salt.
      // PasswordDeriveBytes -- It Derives a key from a HashValue
      PasswordDeriveBytes SecretKey = new PasswordDeriveBytes(HashValue, Salt);

      // Create a encryptor from the existing SecretKey bytes.
      // We use 32 bytes for the secret key
      // (the default Rijndael key length is 256 bit = 32 bytes) and
      // then 16 bytes for the IV (initialization vector),
      // (the default Rijndael IV length is 128 bit = 16 bytes)
      ICryptoTransform Encryptor = RijndaelCipher.CreateEncryptor(SecretKey.GetBytes(16), SecretKey.GetBytes(16));

      // Create a MemoryStream that is going to hold the encrypted bytes
      MemoryStream memoryStream = new MemoryStream();

      // Create a CryptoStream through which we are going to be processing our data.
      // CryptoStreamMode.Write means that we are going to be writing data
      // to the stream and the output will be written in the MemoryStream
      // we have provided. (always use write mode for encryption)
      CryptoStream cryptoStream = new CryptoStream(memoryStream, Encryptor, CryptoStreamMode.Write);

      // Start the encryption process.
      cryptoStream.Write(PlainText, 0, PlainText.Length);

      // Finish encrypting.
      cryptoStream.FlushFinalBlock();

      // Convert our encrypted data from a memoryStream into a byte array.
      byte[] CipherBytes = memoryStream.ToArray();

      // Close both streams.
      memoryStream.Close();
      cryptoStream.Close();
      // Convert encrypted data into a base64-encoded string.
      // A common mistake would be to use an Encoding class for that.
      // It does not work, because not all byte values can be
      // represented by characters. We are going to be using Base64 encoding
      // That is designed exactly for what we are trying to do.
      string EncryptedData = Convert.ToBase64String(CipherBytes);

      RijndaelCipher.Clear();

      // Return encrypted string.
      return EncryptedData;
    }

    public string DecryptString(string InputText, string Password)
    {
      try
      {
        RijndaelManaged RijndaelCipher = new RijndaelManaged();

        byte[] EncryptedData = Convert.FromBase64String(InputText);
        byte[] Salt = Encoding.ASCII.GetBytes(Password.Length.ToString());

        PasswordDeriveBytes SecretKey = new PasswordDeriveBytes(Password, Salt);

        // Create a decryptor from the existing SecretKey bytes.
        ICryptoTransform Decryptor = RijndaelCipher.CreateDecryptor(SecretKey.GetBytes(16), SecretKey.GetBytes(16));
        MemoryStream memoryStream = new MemoryStream(EncryptedData);

        // Create a CryptoStream. (always use Read mode for decryption).
        CryptoStream cryptoStream = new CryptoStream(memoryStream, Decryptor, CryptoStreamMode.Read);

        // Since at this point we don't know what the size of decrypted data
        // will be, allocate the buffer long enough to hold EncryptedData;
        // DecryptedData is never longer than EncryptedData.
        byte[] PlainText = new byte[EncryptedData.Length];

        // Start decrypting.
        int DecryptedCount = cryptoStream.Read(PlainText, 0, PlainText.Length);

        memoryStream.Close();
        cryptoStream.Close();

        // Convert decrypted data into a string.
        string DecryptedData = Encoding.Unicode.GetString(PlainText, 0, DecryptedCount);

        RijndaelCipher.Clear();

        return DecryptedData;
      }
      catch(Exception e)
      {
        using (StreamWriter sw = new StreamWriter(@"C:\Users\CFC\Desktop\error.txt"))
        {
          sw.WriteLine(e.Message);
          sw.WriteLine(e.InnerException);
          sw.WriteLine(e.Source);
          sw.WriteLine(e.StackTrace);
        }

        return null;
      }
    }
  }
}
```

APPENDIX VIII

C# CODE: CLASS RECOVERY SCHEME

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;
using Security;
using Ionic.Zip;
using ProjectHelper;
using System.IO;

namespace Recovery
{
  public class RecoveryScheme
  {
    private string recoveryContentString;

    public RecoveryScheme()
    {
      recoveryContentString = string.Empty;
    }

    public string createRecoveryFile(string imageString, string watermarkString, string Hash, string Salt, string onwerInfo,
string watermarkProperties, bool isImage)
    {
      byte[] imageBytes = Convert.FromBase64String(imageString);

      string imageHash = calculateSHA512Hash(imageBytes);
      string watermarkHash;

      if(isImage)
        watermarkHash = calculateSHA512Hash(Convert.FromBase64String(watermarkString));
      else
        watermarkHash = calculateSHA512Hash(Encoding.ASCII.GetBytes(watermarkString));

      string recoveryContent = string.Format("{0}//;//{1}//;//{2}//;//{3}//;//{4}//;//{5}", Hash, Salt, imageString,
watermarkProperties, onwerInfo, watermarkString);

      Hashing _hashObj = new Hashing();
      EncryptionandDecryption _EncandDecObj1 = new EncryptionandDecryption();

      string recoveryHash;
      string recoverySalt;

      string password = calculateSHA512Hash(Encoding.ASCII.GetBytes(recoveryContent));

      _hashObj.GetHashAndSaltString(password, out recoveryHash, out recoverySalt);
      string encPass = _hashObj.GetHash(recoveryHash, recoverySalt);

      string encryptedRecoveryContent = _EncandDecObj1.EncryptString(recoveryContent,encPass);

      string finalContent = string.Format("{0}{1}{2}{3}{4}", imageHash, watermarkHash, recoveryHash, recoverySalt,
encryptedRecoveryContent);

      return finalContent;
    }

    private string calculateSHA512Hash(byte [] imageBytes)
    {
      SHA512 sha512 = SHA512CryptoServiceProvider.Create();
      byte[] hash = sha512.ComputeHash(imageBytes);
    return Convert.ToBase64String(hash);
}

private bool compareTwoHash(byte[] imageBytes, string hashString)
{
    string hashValue = calculateSHA512Hash(imageBytes);

    if (string.Equals(hashValue, hashString))
        return true;
    else
        return false;
}

public bool compareHash(string filePath, byte[] imageBytes, bool isImage)
{
    var recoveryStream = new MemoryStream();

    using (ZipFile zip = ZipFile.Read(filePath))
    {
        ZipEntry imentry = zip[ConstantHelper.AppsSetting.RECOVERY];
        imentry.Extract(recoveryStream);
    }

    recoveryContentString = Encoding.ASCII.GetString(recoveryStream.ToArray());

    string Hash;

    if(isImage)
        Hash = recoveryContentString.Substring(0, 88);
    else
        Hash = recoveryContentString.Substring(88, 88);

    return compareTwoHash(imageBytes, Hash);
}

public string performRecovery(string filepath, string fileName, bool isImage)
{
    Hashing _hashObj = new Hashing();
    EncryptionandDecryption _EncandDecObj1 = new EncryptionandDecryption();
    EncryptionandDecryption _EncandDecObj2 = new EncryptionandDecryption();

    string recoveryHash = recoveryContentString.Substring(176, 88);
    string recoverySalt = recoveryContentString.Substring(264, 12);

    string encPass = _hashObj.GetHash(recoveryHash, recoverySalt);
    string encryptedRecoveryContent = recoveryContentString.Substring(276);

    string decryptedRecoveryContent = _EncandDecObj1.DecryptString(encryptedRecoveryContent, encPass);

    string [] separator = {"//;//"};
    string [] recoveryContent = decryptedRecoveryContent.Split(separator, 6, StringSplitOptions.None);

    string Hash = recoveryContent[0];
    string Salt = recoveryContent[1];

    string fileContent = string.Empty;
    string returnString = string.Empty;

    if (isImage)
    {
        returnString = recoveryContent[2];

        string encPassImage = _hashObj.GetHash(Hash, Salt);
        string encryptedImageString = _EncandDecObj1.EncryptString(recoveryContent[2], encPassImage);
```

```
            string header = Hash + Salt;

            fileContent = header + encryptedImageString;
          }
          else
          {
            returnString = recoveryContent[3];

            string wmPass = _hashObj.GetHash(Salt, Hash);

            string encryptedOwner = _EncandDecObj1.EncryptString(recoveryContent[4], wmPass);
            string encryptedWatermarkString = _EncandDecObj2.EncryptString(recoveryContent[5], wmPass);

            fileContent = string.Format("{0}//;//{1}//;//{2}", recoveryContent[3], encryptedOwner, encryptedWatermarkString);
          }

          using (ZipFile zip = ZipFile.Read(filepath))
          {
            zip.UpdateEntry(fileName, fileContent);
            zip.CompressionLevel = Ionic.Zlib.CompressionLevel.BestCompression;
            zip.Save();
          }

          return returnString;

        }
      }
    }
```